Test-case Generator for Nonlinear Continuous Parameter Optimization Techniques

Zbigniew Michalewicz* , Kalyanmoy Deb† , Martin Schmidt‡, and Thomas Stidsen§

Abstract

The experimental results reported in many papers suggest that making an appropriate a priori choice of an evolutionary method for a nonlinear parameter optimization problem remains an open question. It seems that the most promising approach at this stage of research is experimental, involving a design of a scalable *test suite of constrained optimization problems, in which many features could be easily tuned. Then it would be possible to evaluate merits and drawbacks of the available methods as well as test new methods efficiently.*

In this paper we propose such a test-case generator for constrained parameter optimization techniques. This generator is capable of creating various test problems with different characteristics, like (1) problems with different relative size of the feasible region in the search space; (2) problems with different number and types of constraints; (3) problems with convex or non-convex objective function, possibly with multiple optima; (4) problems with highly non-convex constraints consisting of (possibly) disjoint regions. Such a test-case generator is very useful for analyzing and comparing different constraint-handling techniques.

Keywords: *evolutionary computation, nonlinear programming, constrained optimization, test case generator*

1 Introduction

The general nonlinear programming (NLP) problem is to find $\vec{x}$ so as to

$$\text{optimize } f(\vec{x}), \quad \vec{x} = (x_1, \ldots, x_n) \in I\!R^n, \tag{1}$$

where $\vec{x} \in \mathcal{F} \subseteq \mathcal{S}$. The *objective function* f is defined on the *search space* $\mathcal{S} \subseteq I\!R^n$ and the set $\mathcal{F} \subseteq \mathcal{S}$ defines the *feasible region*. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in $I\!R^n$ (domains of variables defined by their lower and upper bounds):

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n,$$

*Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA *and* at the Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland, e-mail: *zbyszek@uncc.edu*

†Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, Pin 208 016, India, e-mail: *deb@iitk.ac.in*

‡Department of Computer Science, Aarhus University, DK-8000 Aarhus C, Denmark, e-mail: *marsch@daimi.au.dk*

§Department of Computer Science, Aarhus University, DK-8000 Aarhus C, Denmark, e-mail: *stidsen@daimi.au.dk*

whereas the feasible region $\mathcal{F} \subseteq \mathcal{S}$ is defined by a set of p additional constraints ($p \geq 0$):

$$g_j(\vec{x}) \leq 0, \text{ for } j = 1, \ldots, q, \text{ and } h_j(\vec{x}) = 0, \text{ for } j = q+1, \ldots, p.$$

At any point $\vec{x} \in \mathcal{F}$, the constraints g_j that satisfy $g_j(\vec{x}) = 0$ are called the *active* constraints at $\vec{x}$.

The NLP problem, in general, is intractable: it is impossible to develop a deterministic method for the NLP in the global optimization category, which would be better than the exhaustive search (Gregory, 1995). This makes a room for evolutionary algorithms, extended by some constraint-handling methods. Indeed, during the last few years, several evolutionary algorithms (which aim at complex objective functions (e.g., non differentiable or discontinuous) have been proposed for the NLP; a recent survey paper (Michalewicz and Schoenauer 1996) provides an overview of these algorithms.

It is not clear what characteristics of a constrained problem make it difficult for an evolutionary technique (and, as a matter of fact, for any other optimization technique). Any problem can be characterized by various parameters; these may include the number of linear constraints, the number of nonlinear constraints, the number of equality constraints, the number of active constraints, the ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ of sizes of feasible search space to the whole, the type of the objective function (the number of variables, the number of local optima, the existence of derivatives, etc). In (Michalewicz and Schoenauer 1996) eleven test cases for constrained numerical optimization problems were proposed ($G1$–$G11$). These test cases include objective functions of various types (linear, quadratic, cubic, polynomial, nonlinear) with various number of variables and different types (linear inequalities, nonlinear equations and inequalities) and numbers of constraints. The ratio ρ between the size of the feasible search space $\mathcal{F}$ and the size of the whole search space $\mathcal{S}$ for these test cases vary from 0% to almost 100%; the topologies of feasible search spaces are also quite different. These test cases are summarized in Table 1. For each test case the number n of variables, type of the function f, the relative size of the feasible region in the search space given by the ratio ρ, the number of constraints of each category (linear inequalities *LI*, nonlinear equations *NE* and inequalities *NI*), and the number a of active constraints at the optimum (including equality constraints) are listed.

The results of many tests did not provide meaningful conclusions, as no single parameter (number of linear, nonlinear, active constraints, the ratio ρ, type of the function, number of variables) proved to be significant as a major measure of difficulty of the problem. For example, many tested methods approached the optimum quite closely for the test cases $G1$ and $G7$ (with $\rho = 0.0111\%$ and $\rho = 0.0003\%$, respectively), whereas most of the methods experienced difficulties for the test case $G10$ (with $\rho = 0.0010\%$). Two quadratic functions (the test cases $G1$ and $G7$) with a similar number of constraints (9 and 8, respectively) and an identical number (6) of active constraints at the optimum, gave a different challenge to most of these methods. Also, several methods were quite sensitive to the presence of a feasible solution in the initial population. Possibly a more extensive testing of various methods was required.

Not surprisingly, the experimental results of (Michalewicz and Schoenauer 1996) suggested that making an appropriate *a priori* choice of an evolutionary method for a nonlinear optimization problem remained an open question. It seems that more complex properties of the problem (e.g., the characteristic of the objective function together with the topology of the feasible region) may constitute quite significant measures of the difficulty of the problem. Also, some additional measures of the problem characteristics due to the constraints might be helpful. However, this kind of information is not generally available. In (Michalewicz and Schoenauer 1996) the authors wrote:

> "It seems that the most promising approach at this stage of research is experimental, involving the design of a scalable test suite of constrained optimization problems, in

Function	n	Type of f	ρ	LI	NE	NI	a
$G1$	13	quadratic	0.0111%	9	0	0	6
$G2$	k	nonlinear	99.8474%	0	0	2	1
$G3$	k	polynomial	0.0000%	0	1	0	1
$G4$	5	quadratic	52.1230%	0	0	6	2
$G5$	4	cubic	0.0000%	2	3	0	3
$G6$	2	cubic	0.0066%	0	0	2	2
$G7$	10	quadratic	0.0003%	3	0	5	6
$G8$	2	nonlinear	0.8560%	0	0	2	0
$G9$	7	polynomial	0.5121%	0	0	4	2
$G10$	8	linear	0.0010%	3	0	3	6
$G11$	2	quadratic	0.0000%	0	1	0	1

Table 1: Summary of eleven test cases. The ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ was determined experimentally by generating 1,000,000 random points from $\mathcal{S}$ and checking whether they belong to $\mathcal{F}$ (for $G2$ and $G3$ we assumed $k = 50$). *LI, NE*, and *NI* represent the number of linear inequalities, and nonlinear equations and inequalities, respectively

> which many [...] features could be easily tuned. Then it should be possible to test new methods with respect to the corpus of all available methods."

Clearly, there is a need for a parameterized test-case generator which can be used for analyzing various methods in a systematic way (rather than testing them on a few selected test cases; moreover, it is not clear whether addition of a few extra test cases is of any help).

In this paper we propose such a test-case generator for constrained parameter optimization techniques. This generator is capable of creating various test cases with different characteristics:

- problems with different value of ρ: the relative size of the feasible region in the search space;
- problems with different number and types of constraints;
- problems with convex or non-convex objective function, possibly with multiple optima;
- problems with highly non-convex constraints consisting of (possibly) disjoint regions.

All this can be achieved by setting a few parameters which influence different characteristics of the optimization problem. Such test-case generator should be very useful for analyzing and comparing different constraint-handling techniques.

There were some attempts in the past to propose a test case generator for unconstrained parameter optimization (Whitley et al. 1995; Whitley et al. 1996). We are also aware of one attempt to do so for constrained cases; in (Kemenade 1998) the author proposed so-called stepping-stones problem defined as:

objective: maximize $\sum_{i=1}^{n}(x_i/\pi + 1)$,

where $-\pi \leq x_i \leq \pi$ for $i = 1, \ldots, n$ and the following constraints are satisfied:

$e^{x_i/\pi} + cos(2x_i) \leq 1$ for $i = 1, \ldots, n$.

Note that the objective function is linear and that the feasible region is split into 2^n disjoint parts (called stepping-stones). As the number of dimensions n grows, the problem becomes more complex. However, as the stepping-stones problem has one parameter only, it can not be used to investigate some aspects of a constraint-handling method. In (Michalewicz et al., 1999a; Michalewicz et al., 1999b) we reported on preliminary experiments with a test case generator.

The paper is organized as follows. The following section describes the proposed test-case generator for constrained parameter optimization techniques. Section 3 surveys briefly several constraint-handling techniques for numerical optimization problems which have emerged in evolutionary computation techniques over the last years. Section 4 discusses experimental results of one particular constraint-handling technique on a few generated test cases. Section 5 concludes the paper and indicates some directions for future research.

2 Test-case generator

As explained in the Introduction, it is of great importance to have a parameterized generator of test cases for constrained parameter optimization problems. By changing values of some parameters it would be possible to investigate merits/drawbacks (efficiency, cost, etc) of many constraint-handling methods. Many interesting questions could be addressed:

- how the efficiency of a constraint-handling method changes as a function of the number of disjoint components of the feasible part of the search space?
- how the efficiency of a constraint-handling method changes as a function of the ratio between the sizes of the feasible part and the whole search space?
- what is the relationship between the number of constraints (or the number of dimensions, for example) of a problem and the computational effort of a method?

and many others. In the following part of this section we describe such parameterized test-case generator

$\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$;

the meaning of its six parameters is as follows:

n	–	the number of variables of the problem
w	–	a parameter to control the number of optima in the search space
λ	–	a parameter to control the number of constraints (inequalities)
α	–	a parameter to control the connectedness of the feasible search regions
β	–	a parameter to control the ratio of the feasible to total search space
μ	–	a parameter to influence the ruggedness of the fitness landscape

The following subsection explains the general ideas behind the proposed concepts. Subsection 2.2 describes some details of the test-case generator $\mathcal{TCG}$, and subsection 2.3 graphs a few landscapes. Subsection 2.4 discusses further enhancements incorporated in the $\mathcal{TCG}$, and subsection 2.5 summarizes some of its properties.

2.1 Preliminaries

The general idea is to divide the search space $\mathcal{S}$ into a number of disjoint subspaces $\mathcal{S}_k$ and to define a unimodal function f_k for every $\mathcal{S}_k$. Thus the objective function G is defined on $\mathcal{S}$ as follows:

$$G(\vec{x}) = f_k(\vec{x}) \text{ iff } \vec{x} \in \mathcal{S}_k.$$

The number of subspaces $\mathcal{S}_k$ corresponds to the total number of local optima of function G.

Each subspace $\mathcal{S}_k$ is divided further into its feasible $\mathcal{F}_k$ and infeasible $\mathcal{I}_k$ parts; it may happen, that one of these parts is empty. This division of $\mathcal{S}_k$ is obtained by introduction of a double inequality which feasible points must satisfy. The feasible part $\mathcal{F}$ of the whole search space $\mathcal{S}$ is defined then as a union of all $\mathcal{F}_k$'s.

The final issue addressed in the proposed model concerns the relative heights of local optima of functions f_k. The global optimum is always located in $\mathcal{S}_0$, but the heights of other peaks (i.e., local optima) may determine whether the problem would be easier or harder to solve.

Let us discuss now the connections between the above ideas and the parameters of the test-case generator. The first (integer) parameter n of the $\mathcal{TCG}$ determines the number of variables of the problem; clearly, $n \geq 1$. The next (integer) parameter $w \geq 1$ determines the number of local optima in the search space, as the search space $\mathcal{S}$ is divided into w^n disjoint subspaces $\mathcal{S}_k$ $(0 \leq k \leq w^n - 1)$ and there is a unique unconstrained optimum in each $\mathcal{S}_k$. The subspaces $\mathcal{S}_k$ are defined in section 2.2; however, the idea is to divide the domain of each of n variables into w disjoint and equal sized segments of the length $\frac{1}{w}$. The parameter w determines also the boundaries of the domains of all variables: for all $1 \leq i \leq n$, $x_i \in [-\frac{1}{2w},\ 1 - \frac{1}{2w})$. Thus the search space $\mathcal{S}$ is defined as a n-dimensional rectangle:[1]

$$\mathcal{S} = \prod_{i=1}^{n} [-\tfrac{1}{2w},\ 1 - \tfrac{1}{2w});$$

consequently $|\mathcal{S}| = 1$.

The third parameter λ of the test-case generator is related to the number m of constraints of the problem, as the feasible part $\mathcal{F}$ of the search space $\mathcal{S}$ is defined by means on m double inequalities, called "rings" or "hyper-spheres", $(1 \leq m \leq w^n)$:

$$r_1^2 \leq c_k(\vec{x}) \leq r_2^2, \quad k = 0, \ldots, m-1, \tag{2}$$

where $0 \leq r_1 \leq r_2$ and each $c_k(\vec{x})$ is a quadratic function:

$$c_k(\vec{x}) = (x_1 - p_1^k)^2 + \ldots + (x_n - p_n^k)^2,$$

where $(p_1^k, \ldots, p_n^k)$ is the center of the kth ring.

These m double inequalities define m feasible parts $\mathcal{F}_k$ of the search space:

$$\vec{x} \in \mathcal{F}_k \text{ iff } r_1^2 \leq c_k(\vec{x}) \leq r_2^2,$$

and the overall feasible search space $\mathcal{F} = \cup_{k=0}^{m-1} \mathcal{F}_k$. Note, the interpretation of constraints here is different than the one in the standard definition of the NLP problem (see Equation (1)): Here the search space $\mathcal{F}$ is defined as a *union* (not intersection) of all double constraints. In other words, a

[1] In section 2.4 we define an additional transformation from $[-\frac{1}{2w},\ 1 - \frac{1}{2w})$ to $[0, 1)$ to simplify the usability of the $\mathcal{TCG}$.

point $\vec{x}$ is feasible if and only if there exist an index $0 \le k \le m-1$ such that double inequality in Equation (2) is satisfied. Note also, that if $m < w^n$, then $\mathcal{F}_k$ are empty for all $m \le k \le w^n - 1$.

The parameter $0 \le \lambda \le 1$ determines the number m of constraints as follows:

$$m = \lfloor \lambda(w^n - 1) + 1 \rfloor. \tag{3}$$

Clearly, $\lambda = 0$ and $\lambda = 1$ imply $m = 1$ and $m = w^n$, i.e., minimum and maximum number of constraints, respectively.

There is one important implementational issue connected with a representation of the number w^n. This number might be too large to store as an integer variable in a program (e.g., for $w = 10$ and $n = 500$); consequently the value of m might be too large as well. Because of that the Equation (3) should be interpreted as the expected value of random variable m rather than the exact formula. This is achieved as follows. Note that an index k of a subspace $\mathcal{S}_k$ can be represented in a n-dimensional w-ary alphabet (for $w > 1$) as $(q_{1,k}, \ldots, q_{n,k})$, i.e.,

$k = \sum_{i=1}^{n} q_{i,k} w^{n-i}$.

Then, for each dimension, we (randomly) select a fraction $(\lambda)^{\frac{1}{n}}$ of indices; the subspace $\mathcal{S}_k$ contains one of m constraints iff an index $q_{i,k}$ was selected for dimension i $(1 \le i \le n)$. Note that the probability of selecting any subspace $\mathcal{S}_k$ is λ. For the same reason later in the paper (see sections 2.2 and 2.5), new parameters (k' and k'') replace the original k (e.g., Equations (21) and (28)). The center $(p_1^k, \ldots, p_n^k)$ of a hyper-sphere defined by a particular c_k $(0 \le k \le m-1)$ is determined as follows:

$$(p_1^k, \ldots, p_n^k) = (q_{n,k}/w, \ldots, q_{1,k}/w), \tag{4}$$

where $(q_{1,k}, \ldots, q_{n,k})$ is a n-dimensional representation of the number k in w-ary alphabet.[2]

Let us illustrate concepts introduced so far by the following example. Assume $n = 2$, $w = 5$, and $m = 22$ (note that $m \le w^n$). Let us assume that r_1 and r_2 (smaller and larger radii of all hyper-spheres (circles in this case), respectively) have the following values:[3]

$r_1 = 0.04$ and $r_2 = 0.09$.

Thus the feasible search space consists of $m = 22$ disjoint rings, and the ratio ρ between the sizes of the feasible part $\mathcal{F}$ and the whole search space $\mathcal{S}$ is 0.449. The search space $\mathcal{S}$ and the feasible part $\mathcal{F}$ are displayed in Figure 1(a).

Note that the center of the "first" sphere defined by c_0 (i.e., $k = 0$) is $(p_1^0, p_2^0) = (0, 0)$, as $k = 0 = (0, 0)_5$. Similarly, the center of the "fourteenth" sphere (out of $m = 22$) defined by c_{13} (i.e., $k = 13$) is $(p_1^{13}, p_2^{13}) = (3/w, 2/w)$, as $k = 13 = (2, 3)_5$.

With $r_2 = 0$ (which also implies $r_1 = 0$), the feasible search space would consist of a set of m points. It is interesting to note that for $r_1 = 0$:

- if $0 \le r_2 < \frac{1}{2w}$, the feasible search space $\mathcal{F}$ consists of m disjoint convex components; however the overall search space is non-convex;

[2]We assumed here, that $w > 1$; for $w = 1$ there is no distinction between the search space $\mathcal{S}$ and the subspace $\mathcal{S}_0$, and consequently m is one (a single double constraint). In the latter case, the center of the only hyper-sphere is the center of the search space.

[3]These values are determined by two other parameters of the test-case generator, α and β, as discussed later.

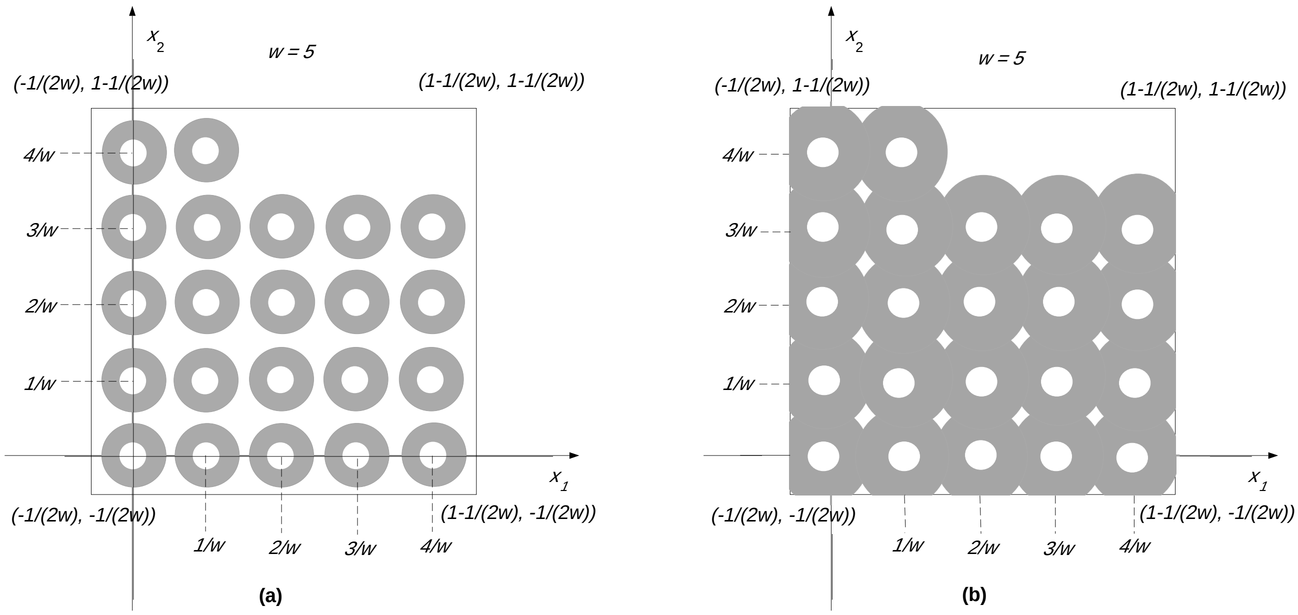


Figure 1: The search spaces $\mathcal{S}$ and their feasible parts $\mathcal{F}$ (shaded areas) for test cases with $n = 2$, $m = 22$, $r_1 = 0.04$, and (a) $r_2 = 0.09$ or (b) $r_2 = 0.12$

- if $\frac{1}{2w} \leq r_2 < \frac{\sqrt{n}}{2w}$, the feasible search space $\mathcal{F}$ consists of one connected component; however, this component is highly non-convex with many "holes" among spheres (see Figure 2(a));
- if $r_2 = \frac{\sqrt{n}}{2w}$, most of the whole search space would be feasible (except the "right-top corner" of the search space due to the fact that the number of spheres m might be smaller than w^n; see Figure 2(b)). In any case, $\mathcal{F}$ consists of a single connected component.[4]

Based on the above discussions, we can assume that radii r_1 and r_2 satisfy the following inequalities:

$$0 \leq r_1 \leq r_2 \leq \frac{\sqrt{n}}{2w}. \tag{5}$$

Now we can define the fourth and fifth control parameters of the test-case generator $\mathcal{TCG}$:

$$\alpha = \frac{2wr_2}{\sqrt{n}}, \tag{6}$$

$$\beta = \frac{r_1}{r_2}. \tag{7}$$

The operating range for α and β is $0 \leq \alpha, \beta \leq 1$. In that way, the radii r_1 and r_2 are defined by the parameters of the test-case generator:

$$r_1 = \frac{\alpha\beta\sqrt{n}}{2w}, \quad r_2 = \frac{\alpha\sqrt{n}}{2w}. \tag{8}$$

Note that if $\alpha < 1/\sqrt{n}$, the feasible 'islands' $\mathcal{F}_k$ are not connected, as discussed above. On the other hand, for $\alpha \geq 1/\sqrt{n}$, the feasible subspaces $\mathcal{F}_k$ are connected. Thus, the parameter α controls the connectedness of the feasible search space.[5]

[4] Note that the exact value of ρ depends on m: if $m = w^n$, then $\rho = 1$; otherwise, $\rho < 1$.

[5] We shall see later, that this parameter, together with parameter β, also controls the amount of deviation of the constrained maximum solution from the unconstrained maximum solution.

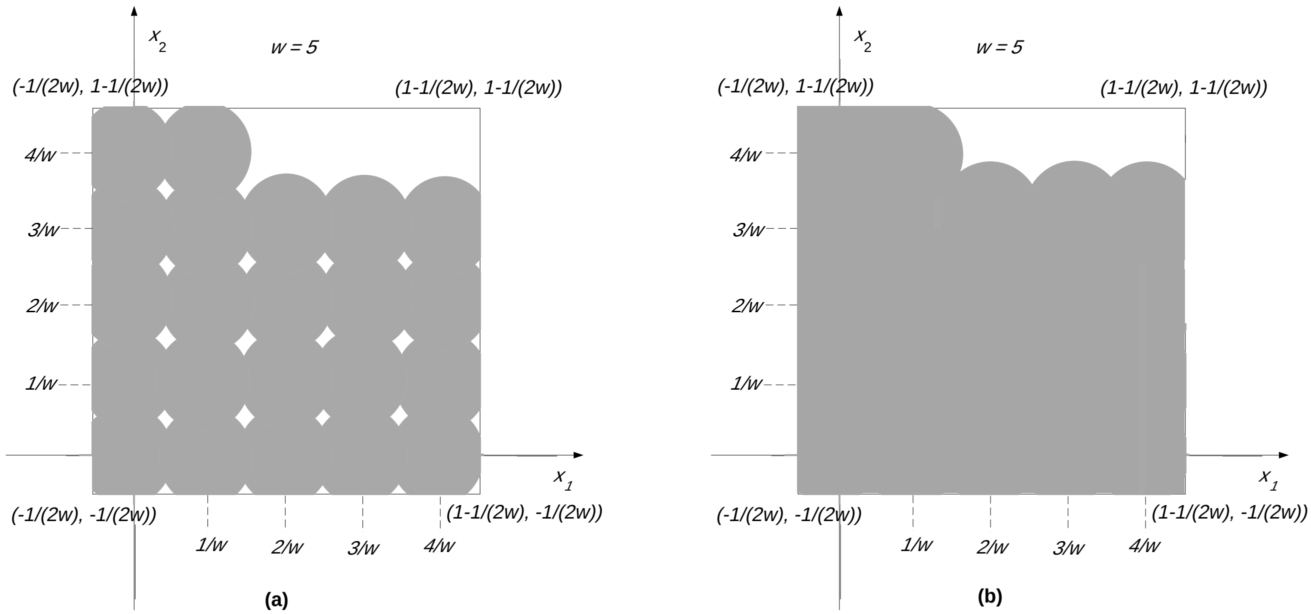


Figure 2: The search spaces $\mathcal{S}$ and their feasible parts $\mathcal{F}$ (shaded areas) for test cases with $n = 2$, $m = 22$, $r_1 = 0$, and (a) $r_2 = 0.12$ or (b) $r_2 = \frac{\sqrt{2}}{2w} = \frac{\sqrt{2}}{10}$

For $\beta = 0$ the feasible parts $\mathcal{F}_k$ are convex and for $\beta > 0$ — always non-convex (see (a) and (b) of Figure 1). If $\beta = 1$ (i.e, $r_1 = r_2$), the feasible search space consists of (possibly parts of) boundaries of spheres; this case corresponds to an optimization problem with equality constraints. The parameter β is also related to the proportion of the ratio ρ of the feasible to the total search space ($\rho = |\mathcal{F}|/|\mathcal{S}|$). For an n-dimensional hyper-sphere, the enclosed volume is given as follows (Rudolph, 1996):

$$V_n(r) = \frac{\pi^{n/2} r^n}{\Gamma(n/2+1)}. \tag{9}$$

When $\alpha \leq 1/\sqrt{n}$, the ratio ρ can be written as follows[6] (note that $|\mathcal{S}| = 1$):

$$\rho = m\left(V_n(r_2) - V_n(r_1)\right) = \frac{(\pi n)^{n/2}\alpha^n}{2^n \Gamma(n/2+1)} \frac{m}{w^n}\left(1 - \beta^n\right). \tag{10}$$

When $\beta \approx 1$ ($\mathcal{F}_k$'s are rings of a small width), the above ratio is

$$\rho \approx \frac{(\pi n)^{n/2}\alpha^n}{2^n \Gamma(n/2+1)} \frac{m}{w^n} n(1-\beta). \tag{11}$$

In the following subsection, we define the test-case generator $\mathcal{TCG}$ in detail; we provide definitions of functions f_k and discuss the sixth parameter μ of this test-case generator.

[6] For $\alpha > 1$, the hyper-spheres overlap and it is not trivial to compute this ratio. However, this ratio gets closer to one as α increases from one.

2.2 Details

As mentioned earlier, the search space $\mathcal{S} = \prod_{i=1}^{n}[-\frac{1}{2w},\ 1-\frac{1}{2w})$ is divided into w^n subspaces $\mathcal{S}_k$, $k = 0, 1, \ldots, (w^n - 1)$. Each subspace $\mathcal{S}_k$ is defined as a n-dimensional cube: $\mathcal{S}_k = \{x_i : l_i^k \leq x_i < u_i^k\}$, where the bounds l_i^k and u_i^k are defined as:

$$l_i^k = \frac{2q_{n-i+1,k} - 1}{2w} \quad \text{and} \quad u_i^k = \frac{2q_{n-i+1,k} + 1}{2w}, \tag{12}$$

for $k = 0, 1, \ldots, (w^n - 1)$. The parameter $q_{n-i+1,k}$ is the $(n-i+1)$-th component of a n-dimensional representation of the number k in w-ary alphabet. For example, the boundaries of the subspace $\mathcal{S}_{13}$ (see Figure 1) are

$\frac{5}{2w} \leq x_1 < \frac{7}{2w}$ and $\frac{3}{2w} \leq x_2 < \frac{5}{2w}$,

as $(2, 3)_5 = 13$, i.e., $(2, 3)$ is a 2-dimensional representation of $k = 13$ in $w = 5$-ary alphabet.

For each subspace $\mathcal{S}_k$, there is a function f_k defined on this subspace as follows:

$$f_k(x_1, \ldots, x_n) = a_k \left(\Pi_{i=1}^{n}(u_i^k - x_i)(x_i - l_i^k)\right)^{\frac{1}{n}}, \tag{13}$$

where a_k's are predefined positive constants. Note that for any $\vec{x} \in \mathcal{S}_k$, $f_k(\vec{x}) \geq 0$; moreover, $f_k(\vec{x}) = 0$ iff $\vec{x}$ is a boundary point of the subspace $\mathcal{S}_k$.

The objective function (to be maximized) of the test-case generator $\mathcal{TCG}$ is defined as follows:

$$G(x_1, \ldots, x_n) = \begin{cases} f_0(x_1, \ldots, x_n) & \text{if } (x_1, \ldots, x_n) \in \mathcal{S}_0 \\ f_1(x_1, \ldots, x_n) & \text{if } (x_1, \ldots, x_n) \in \mathcal{S}_1 \\ . & \\ . & \\ . & \\ f_{w^n-1}(x_1, \ldots, x_n) & \text{if } (x_1, \ldots, x_n) \in \mathcal{S}_{w^n-1}, \end{cases} \tag{14}$$

where

- $-\frac{1}{2w} \leq x_i < 1 - \frac{1}{2w}$ for all $1 \leq i \leq n$, i.e., $\vec{x} = (x_1, \ldots, x_n) \in \mathcal{S}$, and

- $\mathcal{F} = \mathcal{F}_0 \cup \ldots \cup \mathcal{F}_{w^n-1}$, i.e., one of the following m double constraints is satisfied:

  $r_1^2 \leq c_k \leq r_2^2$ $(0 \leq k \leq m-1)$.

Now we are ready to discuss the significance of the predefined constants a_k (Equation (13)). Let us introduce the following notation:

- $(\underline{x}_1^k, \ldots, \underline{x}_n^k)$ – is the maximum solution for *unconstrained* function f_k, i.e., when the constraint $r_1^2 \leq c_k(\vec{x}) \leq r_2^2$ is ignored (with $r_1 > 0$),

- $(\bar{x}_1^k, \ldots, \bar{x}_n^k)$ – is the maximum solution for *constrained* function f_k, i.e., when the constraint $r_1^2 \leq c_k(\vec{x}) \leq r_2^2$ is taken into account.

It is obvious, that the function f_k has its unconstrained maximum at the center of $\mathcal{S}_k$, i.e.,

$\underline{x}_i^k = (l_i^k + u_i^k)/2.$

The corresponding function value at this point is

$$f_k(\underline{x}^k) = \frac{a_k}{4w^2}. \tag{15}$$

Thus, if constraints (with $r_1 > 0$) are not taken into account, the function G has exactly w^n local maxima points: one maximum for each subspace $\mathcal{S}_k$. The global maximum (again, without considering constraints) lies in the subspace $\mathcal{S}_k$, for which the corresponding function f_k has the largest constant a_k (see Equation (13)).

When constraints are taken into account (with $r_1 > 0$) the unconstrained maximum is not feasible anymore. By using first and second-order optimality conditions (Deb, 1995), it can be shown that the new (feasible) maximum at each subspace moves to:

$\bar{x}_i^k = (l_i^k + u_i^k)/2 \pm r_1/\sqrt{n},$

which is located on the inner ring. The corresponding maximum function value in subspace S_k is then

$$f_k(\bar{x}^k) = a_k \cdot (1/(4w^2) - r_1^2/n) = a_k \cdot \left(\frac{1-\alpha^2\beta^2}{4w^2}\right) = \frac{Aa_k}{4w^2}, \tag{16}$$

where $A = 1 - \alpha^2\beta^2$ (the constant A is important in our further discussion). Clearly, for $r_1 > 0$, $A < 1$. For an n-dimensional problem, there are a total of 2^n maxima points in each subspace, each having the same above maximum function value. Thus, there are a total of $(2w)^n$ maxima in the entire search space.

To control the heights of these local maxima, the values of a_k can be arranged in a particular sequence, so that the global maximum solution always occurs in the first subspace (or S_0) and the worst local maximum solution occurs in the subspace S_{w^n-1}. Different sequences of this type result in landscapes of different complexity. This is the role of the sixth parameter μ of the test-case generator $\mathcal{TCG}$: by changing its value we should be able to change the landscape from difficult to easy.

Let us consider first two possible scenarios, which would serve as two extreme cases (i.e., difficult and easy landscapes, respectively) for the parameter μ. In both these cases we assume that $r_1 > 0$ (otherwise, unconstrained maximum overlaps with the constrained one).

Case 1: It seems that a challenging landscape would have the following feature: the constrained maximum solution is no better than the worst local maximum solution of the unconstrained function. This way, if a constraint handling optimizer does not work properly to search in the feasible region, one of many local maximum solutions can be found, instead of the global constrained optimum solution. Realizing that the global maximum solution lies in subspace S_0 and the worst local maximum solutions lies in subspace S_{w^n-1}, we have the following condition:

$$f_0(\bar{x}_1^0, \bar{x}_2^0, \ldots, \bar{x}_n^0) \le f_{w^n-1}(\underline{x}_1^{w^n-1}, \underline{x}_2^{w^n-1}, \ldots, \underline{x}_n^{w^n-1}). \tag{17}$$

Substituting the function values, we obtain the following:

$$\frac{a_{w^n-1}}{a_0} \ge A. \tag{18}$$

Case 2: On the other hand, let us consider an easy landscape, where the constrained maximum solution is no worse than the unconstrained local maximum solution of the next-best subspace. This makes the function easy to optimize, because the purpose is served even if the constraint optimizer is incapable of distinguishing feasible and infeasible regions in the search space, as long as it can find the globally best subspace in the entire search space and concentrate its search there. Note that even in this case there are many local maximum solutions where the optimizer can get stuck to. Thus, this condition tests more an optimizer's ability to find the global maximum solution among many local maximum solutions and does not test too much whether the optimizer can distinguish feasible from infeasible search regions. Mathematically, the following condition must be true:

$$f_0(\bar{x}_1^0, \bar{x}_2^0, \ldots, \bar{x}_n^0) \geq f_1(\underline{x}_1^1, \underline{x}_2^1, \ldots, \underline{x}_n^1), \tag{19}$$

assuming that the next-base subspace is $\mathcal{S}_1$. Substituting the function values, we obtain

$$\frac{a_1}{a_0} \leq A. \tag{20}$$

Thus, to control the degree of difficulty in the function, i.e., to vary between cases 1 and 2, the parameter μ is introduced. It can be adjusted at different values to have the above two conditions as extreme cases. To simplify the matter, we may like to have Condition (18) when μ is set to one and have Condition (20) when μ is set to zero. One such possibility is to have the following term for defining a_k:[7]

$$a_k = (1 - \alpha^2\beta^2)^{(1-\mu')k'} = A^{(1-\mu')k'}. \tag{21}$$

The value of k' is given by the following formula

$$k' = \log_2(\sum_{i=1}^{n} q_{i,k} + 1), \tag{22}$$

where $(q_{1,k}, \ldots, q_{n,k})$ is a n-dimensional representation of the number k in w-ary alphabet. The value of μ' is defined as

$$\mu' = \left(1 - \frac{1}{\log_2(nw - n + 1)}\right)\mu, \tag{23}$$

for $\mu \in [0, 1]$. The term from Equation (21) has the following properties:

1. All a_k are non-negative.
2. The value of a_0 for the first subspace S_0 is always one as $a_0 = A^0 = 1$.
3. The sequence a_k takes $n(w-1)+1$ different values (for $k = 0, \ldots, w^n - 1$).
4. For $\mu = 0$, the sequence $a_k = A^{k'}$ lies in $[A^{\log_2(nw-n+1)}, 1]$ (with $a_0 = 1$ and $a_{w^n-1} = A^{\log_2(nw-n+1)}$). Note that $a_1 = A^1 = A$, so Condition (20) is satisfied. Thus, setting $\mu' = 0$ makes the test function much easier, as discussed above.

[7]Although this term can be used for any $w \geq 1$, we realize that for $w = 1$ there is only one subspace $\mathcal{S}_0$ and a_0 is 1, irrespective of μ. For completeness, we set $\mu = 1$ in this case.

5. For $\mu = 1$, the sequence $a_k = A^{k'/\log_2(nw-n+1)}$ lies in the range $[A, 1]$ (with $a_0 = 1$ and $a_{w^n-1} = A$) and Condition (18) is satisfied with the equality sign. Thus we have Case 1 established which is the most difficult test function of this generator, as discussed above.

Since $\mu = 0$ and $\mu = 1$ are two extreme cases of easiness and difficulty, test functions with various levels of difficulties can be generated by using $\mu \in [0, 1]$. Values of μ closer to one will create more difficult test functions than values closer to zero. In the next section, we show the effect of varying μ in some two-dimensional test functions.

To make the global constrained maximum function value always at 1, we normalize the a_k constants as follows:

$$a'_k = \frac{a_k}{(1/(4w^2) - r_1^2/n)} = \left(\frac{4w^2}{1 - \alpha^2\beta^2}\right) \cdot a_k = 4w^2(1 - \alpha^2\beta^2)^{k'(1-\mu')-1}, \tag{24}$$

where k' and μ' are defined by Equations (22) and (23), respectively. Thus, the proposed test function generator is defined in Equation (14), where the functions f_k are defined in Equation (13). The term a_k in Equation (13) is replaced by a'_k defined in Equation (24). The global maximum solution is at $x_i = \pm r_1/\sqrt{n} = \pm\alpha\beta/(2w)$ for all $i = 1, 2, \ldots, n$ and the function values at these solutions are always equal to one. It is important to note that there are a total 2^n global maximum solutions having the same function value of one and at all these maxima, only one constraint $(c_0(\vec{x}) \geq r_1^2)$ is active.

Let us emphasize again that the above discussion is relevant only for $r_1 > 0$. If $r_1 = 0$, then $A = 1 - \alpha^2\beta^2 = 1$, and all a_k's are equal to one. Thus we need a term defining a_k if $\alpha\beta = 0$ to make peaks of variable heights. In this case we can set

$$a_k = \frac{(\mu - 1)k''}{n(w-1)} + 1, \tag{25}$$

The value of k'' is given by the following formula

$$k'' = \textstyle\sum_{i=1}^{n} q_{i,k},$$

where $(q_{1,k}, \ldots, q_{n,k})$ is a n-dimensional representation of the number k in w-ary alphabet. Thus, for $\mu = 0$ we have $a_0 = 1$ and $a_{w^n-1} = 0$ (the steepest decline of heights of the peaks), whereas for $\mu = 1$, $a_0 = a_1 = \ldots = a_{w^n-1} = 1$ (no decline). To make sure the global maximum value of the constrained function is equal to 1, we can set

$$a'_k = 4w^2 a_k \tag{26}$$

for all k.

2.3 A few examples

In order to get a feel for the objective function at this stage of building the $\mathcal{TCG}$, we plot the surface for different values of control parameters and with $n = 2$ (two variables only). A very simple unimodal test problem can be generated by using the following parameter setting: $\mathcal{TCG}(2, 1, 0, \frac{1}{\sqrt{2}}, 0, 1)$ (see Figure 3). Note that in this case $\beta = 0$ which implies $r_1 = 0$ (see (21)). To show the reducing peak effect we need $w > 1$ and $\beta > 0$, as for $\beta = 0$ all peaks have the same height. Thus we assume

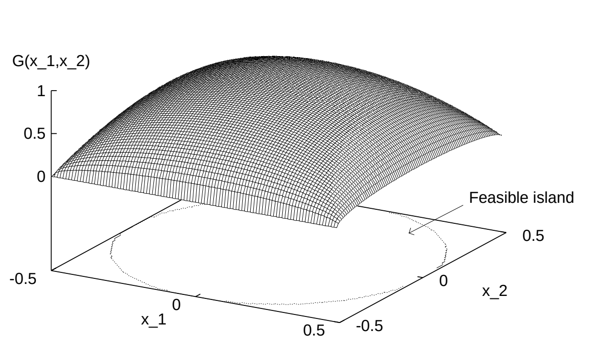


Figure 3: A test case $\mathcal{TCG}(2, 1, 0, \frac{1}{\sqrt{2}}, 0, 1)$

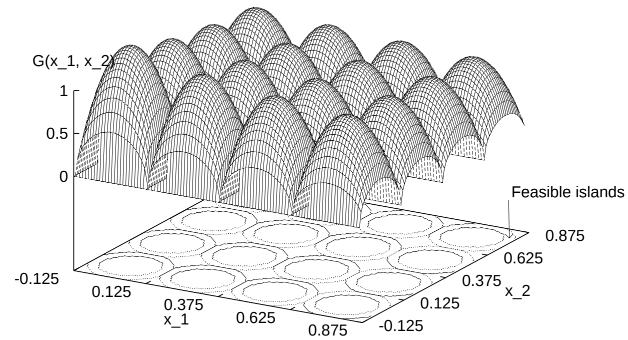


Figure 4: A test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 1)$

$\beta = 0.8$, $w = 4$, and $\lambda = 1$ (i.e., $m = w^n$) in all remaining test cases depicted in the figures of this section.

In Figure 4 we show $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 1.0)$, thereby having $r_2 = 0.125$ and $r_1 = 0.1$. Using Equation (10), we obtain the ratio ρ of feasible to total search space is 0.1875π or 0.589.

Let us discuss the effect of the parameter μ on the shape of the landscape. Figures 4, 5, and 6 are plotted with $w = 4$, but different μ values are used. Recall that the optimal solution always lies in the island closest to the origin. As the μ value is decreased, the feasible island containing the global optimum solution gets more emphasized. The problem depicted in Figure 4 will be more difficult to optimize for global solution than that in Figure 6, because in Figure 4 other feasible islands also contain comparably-good solutions.

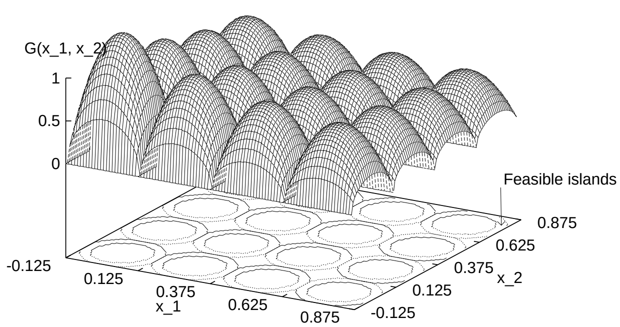


Figure 5: A test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0.5)$

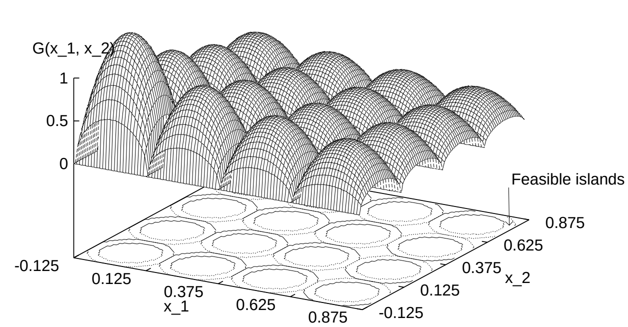


Figure 6: A test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$

Note again, that for a non-zero r_1, the optimal solutions move to the inner boundary (exactly at $x_i = \pm\alpha\beta/(2w)$. Consequently, these problems are harder for optimization algorithms than problems with $r_1 = 0$. A nice aspect of these test functions is that the global feasible function value is always one, no matter what control parameter values are chosen. Figure 7 shows a slice of the function at $x_2 = 0$ for a test problem $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 1.0)$, thus having $r_2 = 0.125$ and $r_1 = 0.1$. Function values corresponding to infeasible solutions are marked using dashed lines. The point 'P' is the globally optimal function value. The infeasible function values (like point ‘Q’) higher than one are also shown. If the constraint handling strategy is not proper, an optimization algorithm may get trapped in local optimal solutions, like ‘Q’, which has better function value but is infeasible.

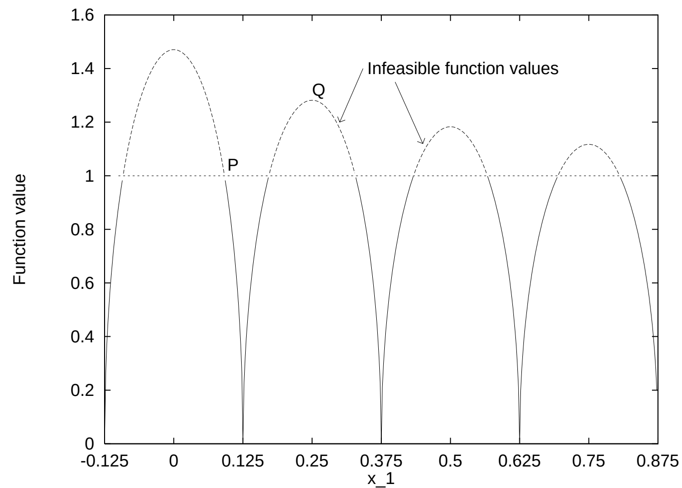


Figure 7: A slice through $x_2 = 0$ of a test problem with $w = 4$, $\mu = 1.0$, and $r_1 = 0.1$ is shown. The global optimum solution is in the left-most attractor

2.4 Further enhancements

As explained in section 2.1, the general idea behind the test-case generator $\mathcal{TCG}$ was to divide the search space $\mathcal{S}$ into a number of disjoint subspaces $\mathcal{S}_k$ and to define an unimodal function f_k for every $\mathcal{S}_k$. The number of subspaces $\mathcal{S}_k$ corresponds to the total number of local optima of function G. To control the heights of these local maxima, the values of a_k were arranged in a particular sequence, so that the global maximum solution always occurs in the first subspace (for $k = 0$, i.e., in S_0) and the worst local maximum solution occurs in the last subspace (for $k = w^n - 1$, i.e., in S_{w^n-1}). Note also that subspaces $\mathcal{S}_k$ of the test-case generator $\mathcal{TCG}$ are arranged in a particular order. For example, for $n = 2$ and $w = 4$, the highest peak is located in subspace $\mathcal{S}_0$, two next-highest peaks (of the same height) are in subspaces $\mathcal{S}_1$ and $\mathcal{S}_4$, three next-highest peaks are in $\mathcal{S}_2$, $\mathcal{S}_5$, $\mathcal{S}_8$, etc. (see Figures 4–6).

To remove this fixed pattern from the generated test cases, we need an additional mechanism: a random permutation of subspaces $\mathcal{S}_k$. Thus we need a procedure *Transform* which randomly permutes indices of subspaces; this is any function

$$\textit{Transform}\colon\ [0..N] \longrightarrow [0..N],$$

such that for any $0 \leq p \neq q \leq N$, $\textit{Transform}(p) \neq \textit{Transform}(q)$. The advantage of such a procedure *Transform* is that we do not need any memory for storing the mapping between indices of subspaces (the size of such memory, for $w > 1$ and large n, may be excessive); however, there is a need to re-compute the permuted index of a subspace $\mathcal{S}_k$ every time we need it.

The procedure *Transform* can be used also to generate a different allocation for m constraints. Note that the $\mathcal{TCG}$ allocates m rings (each defined by a pair of constraints) always to the first m

subspaces $\mathcal{S}_k$, $k = 0, 1, \ldots, m-1$ (see, for example, Figure 1). However, care should be taken always to allocate a ring in the subspace containing the higest peak (thus we know the value of the global feasible, solution).

For example, for $n = 2$ and $w = 4$, our implementation of the procedure *Transform* generated the following permutation of peaks of f_k's:

$$\begin{array}{cccc} 0 \rightarrow 5 & 1 \rightarrow 13 & 2 \rightarrow 9 & 3 \rightarrow 8 \\ 4 \rightarrow 10 & 5 \rightarrow 14 & 6 \rightarrow 15 & 7 \rightarrow 7 \\ 8 \rightarrow 6 & 9 \rightarrow 3 & 10 \rightarrow 0 & 11 \rightarrow 2 \\ 12 \rightarrow 1 & 13 \rightarrow 4 & 14 \rightarrow 12 & 15 \rightarrow 11 \end{array}$$

Figure 8 shows the final outcome (landscape with transformed peaks; see Figure 6 for the original landscape) for the test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$.

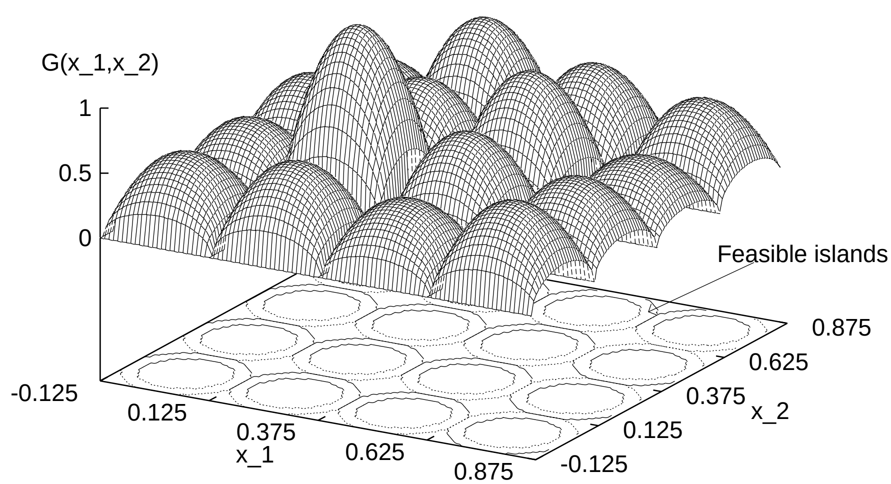


Figure 8: Transformed peaks for the test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$

The final modification of the $\mathcal{TCG}$ maps the domains of parameters x_i from $[-\frac{1}{2w}, 1 - \frac{1}{2w})$ into $[0, 1)$; thus the new search space $\mathcal{S}'$ is

$$\mathcal{S}' = \prod_{i=1}^{n} [0,\ 1),$$

which is more "user-friendly" than the original $\mathcal{S}$. The mapping is a straightforward linear transformation:

$$x_i' \leftarrow x_i + \frac{1}{2w}$$

for all parameters x_i ($i = 1, \ldots, n$). This means that the user supply an input vector $\vec{x} \in [0, 1)^n$, whereas $\mathcal{TCG}$ returns the objective value and a measure of the constraint violation.

2.5 Summary

Based on the above presentations, we now summarize the properties of the test-case generator

$\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$.

The parameters are:

$$\begin{array}{lll} n \geq 1;\text{ integer} & w \geq 1;\text{ integer} & 0 \leq \lambda \leq 1;\text{ float} \\ 0 \leq \alpha \leq 1;\text{ float} & 0 \leq \beta \leq 1;\text{ float} & 0 \leq \mu \leq 1;\text{ float} \end{array}$$

The objective function G is defined by formula (14), where each f_k ($k = 0, \ldots, w^n - 1$) is defined on $\mathcal{S}_k$ as

$$f_k(x'_1, \ldots, x'_n) = a'_k \left(\Pi_{i=1}^n (u_i^k - x_i)(x_i - l_i^k)\right)^{\frac{1}{n}}. \tag{27}$$

All variables x'_i's have the same domain:

$x'_i \in [0, 1)$.

Note, that $x'_i = x_i + \frac{1}{2w}$. The constants a'_k are defined as follows:

$$a'_k = \begin{cases} 4w^2(1 - \alpha^2\beta^2)^{k'[(1-\mu)+\mu/(\log_2(wn-n+1)]-1} & \text{if } \alpha\beta > 0 \\ 4w^2\left(\frac{(\mu-1)k''}{n(w-1)} + 1\right) & \text{if } \alpha\beta = 0, \end{cases} \tag{28}$$

where

$k' = \log_2(\sum_{i=1}^n q_{i,k} + 1)$, and
$k'' = \sum_{i=1}^n q_{i,k}$,

where $(q_{1,k}, \ldots, q_{n,k})$ is a n-dimensional representation of the number k in w-ary alphabet.

If $r_1 > 0$ (i.e., $\alpha\beta > 0$), the function G has 2^n global maxima points, all in permuted $\mathcal{S}_0$. For any global solution $(x_1, \ldots, x_n)$, $x_i = \pm\alpha\beta/(2w)$ for all $i = 1, 2, \ldots, n$. The function values at these solutions are always equal to one.

On the other hand, if $r_1 = 0$ (i.e., $\alpha\beta = 0$), the function G has either one global maximum (if $\mu < 1$) or m maxima points (if $\mu = 1$), one in each of permuted $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$. If $\mu < 1$, the global solution $(x_1, \ldots, x_n)$ is always at

$(x_1, \ldots, x_n) = (0, 0, \ldots, 0)$.

The interpretation of the six parameters of the test-case generator $\mathcal{TCG}$ is as follows:

1. **Dimensionality** n: By increasing the parameter n the dimensionality of the search space can be increased.

2. **Multimodality** w: By increasing the parameter w the multimodality of the search space can be increased. For the unconstrained function, there are w^n local maximum solutions, of which one is globally maximum. For the constrained test function with $\alpha\beta > 0$, there are $(2w)^n$ different local maximum solutions, of which 2^n are globally maximum solutions.

3. **Number of constraints** λ: By increasing the parameter λ the number m of constraints is increased.

4. **Connectedness** α: By reducing the parameter α (from 1 to $1/\sqrt{n}$ and smaller), the connectedness of the feasible subspaces can be reduced. When $\alpha < 1/\sqrt{n}$, the feasible subspaces $\mathcal{F}_k$ are completely disconnected. Additionally, parameter α (with fixed β) influences the proportion of the feasible search space to the complete search space (ratio ρ).

5. **Feasibility** β: By increasing the ratio β the proportion of the feasible search space to the complete search space can be reduced. For β values closer to one, the feasible search space becomes smaller and smaller. These test functions can be used to test an optimizer's ability to be find and maintain feasible solutions.

6. **Ruggedness** μ: By increasing the parameter μ the function ruggedness can be increased (for $\alpha\beta > 0$). A sufficiently rugged function will test an optimizer's ability to search for the globally constrained maximum solution in the presence of other almost equally significant local maxima.

Increasing the each of the above parameters (except α) and decreasing α will cause an increased difficulty for any optimizer. However, it is difficult to conclude which of these factors most profoundly affects the performance of an optimizer. Thus, it is recommended that the user should first test his/her algorithm with the simplest possible combination of the above parameters (small n, small w, small μ, large α, small β, and small λ). Thereafter, the parameters may be changed in a systematic manner to create more difficult test functions. The most difficult test function is created when large values of parameters n, w, λ, β, and μ together with a small value of parameter α are used.

3 Constraint-handling methods

During the last few years several methods were proposed for handling constraints by genetic algorithms for parameter optimization problems. These methods can be grouped into five categories: (1) methods based on preserving feasibility of solutions, (2) methods based on penalty functions, (3) methods which make a clear distinction between feasible and infeasible solutions, (4) methods based on decoders, and (5) other hybrid methods. We discuss them briefly in turn.

3.1 Methods based on preserving feasibility of solutions

The best example of this approach is Genocop (for GEnetic algorithm for Numerical Optimization of COnstrained Problems) system (Michalewicz and Janikow, 1991; Michalewicz et al., 1994). The idea behind the system is based on specialized operators which transform feasible individuals into feasible individuals, i.e., operators, which are closed on the feasible part $\mathcal{F}$ of the search space. The method assumes linear constraints only and a feasible starting point (or feasible initial population). Linear equations are used to eliminate some variables; they are replaced as a linear combination of remaining variables. Linear inequalities are updated accordingly. A closed set of operators maintains feasibility of solutions. For example, when a particular component x_i of a solution vector $\vec{x}$ is mutated, the system determines its current domain $dom(x_i)$ (which is a function of linear constraints and remaining values of the solution vector $\vec{x}$) and the new value of x_i is taken from this domain (either with flat probability distribution for uniform mutation, or other probability distributions for non-uniform and boundary mutations). In any case the offspring solution vector is always feasible. Similarly, arithmetic crossover, $a\vec{x} + (1-a)\vec{y}$, of two feasible solution vectors $\vec{x}$ and $\vec{y}$ yields always a feasible solution (for $0 \leq a \leq 1$) in convex search spaces (the system assumes linear constraints only which imply convexity of the feasible search space $\mathcal{F}$).

Recent work (Michalewicz et al., 1996; Schoenauer and Michalewicz, 1996; Schoenauer and Michalewicz, 1997) on systems which search only the boundary area between feasible and infeasible regions of the search space, constitutes another example of the approach based on preserving feasibility of solutions. These systems are based on specialized boundary operators (e.g., sphere crossover, geometrical crossover, etc.): it is a common situation for many constrained optimization problems that some constraints are active at the target global optimum, thus the optimum lies on the boundary of the feasible space.

3.2 Methods based on penalty functions

Many evolutionary algorithms incorporate a constraint-handling method based on the concept of (exterior) penalty functions, which penalize infeasible solutions. Usually, the penalty function is based on the distance of a solution from the feasible region $\mathcal{F}$, or on the effort to "repair" the solution, i.e., to force it into $\mathcal{F}$. The former case is the most popular one; in many methods a set of functions f_j $(1 \leq j \leq m)$ is used to construct the penalty, where the function f_j measures the violation of the j-th constraint in the following way:

$$f_j(\vec{x}) = \begin{cases} \max\{0, g_j(\vec{x})\}, & \text{if } 1 \leq j \leq q \\ |h_j(\vec{x})|, & \text{if } q+1 \leq j \leq m. \end{cases}$$

However, these methods differ in many important details, how the penalty function is designed and applied to infeasible solutions. For example, a method of static penalties was proposed (Homaifar et al., 1994); it assumes that for every constraint we establish a family of intervals which determine appropriate penalty coefficient. The method of dynamic penalties was examined (Joines and Houck, 1994), where individuals are evaluated (at the iteration t) by the following formula:

$$eval(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\vec{x}),$$

where C, α and β are constants. Another approach (Genocop II), also based on dynamic penalties, was described (Michalewicz and Attia, 1994). In that algorithm, at every iteration active constraints only are considered, and the pressure on infeasible solutions is increased due to the decreasing values of temperature τ. In (Eiben and Ruttkay, 1996) a method for solving constraint satisfaction problems that changes the evaluation function based on the performance of a EA run was described: the penalties (weights) of those constraints which are violated by the best individual after termination are raised, and the new weights are used in the next run. A method based on adaptive penalty functions was developed in (Bean and Hadj-Alouane, 1992; Hadj-Alouane and Bean, 1992): one component of the penalty function takes a feedback from the search process. Each individual is evaluated by the formula:

$$eval(\vec{x}) = f(\vec{x}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\vec{x}),$$

where $\lambda(t)$ is updated every generation t with respect to the current state of the search (based on last k generations). The adaptive penalty function was also used in (Smith and Tate, 1993), where both the search length and constraint severity feedback was incorporated. It involves the estimation of a near-feasible threshold q_j for each constraint $1 \leq j \leq m$); such thresholds indicate distances from the feasible region $\mathcal{F}$ which are "reasonable" (or, in other words, which determine "interesting" infeasible solutions, i.e., solutions relatively close to the feasible region). An additional method (so-called segregated genetic algorithm) was proposed in (Leriche et al., 1995) as yet another way to handle the problem of the robustness of the penalty level: two different penalized fitness functions with static penalty terms p_1 and p_2 were designed (smaller and larger, respectively). The main idea is that such an approach will result roughly in maintaining two subpopulations: the individuals selected on the basis of f_1 will more likely lie in the infeasible region while the ones selected on the basis of f_2 will probably stay in the feasible region; the overall process is thus allowed to reach the feasible optimum from both sides of the boundary of the feasible region.

3.3 Methods based on a search for feasible solutions

There are a few methods which emphasize the distinction between feasible and infeasible solutions in the search space $\mathcal{S}$. One method, proposed in (Schoenauer and Xanthakis, 1993) (called a "behavioral memory" approach) considers the problem constraints in a sequence; a switch from one constraint to another is made upon arrival of a sufficient number of feasible individuals in the population.

The second method, developed in (Powell and Skolnick, 1993) is based on a classical penalty approach with one notable exception. Each individual is evaluated by the formula:

$$eval(\vec{x}) = f(\vec{x}) + r \sum_{j=1}^{m} f_j(\vec{x}) + \theta(t, \vec{x}),$$

where r is a constant; however, the original component $\theta(t, \vec{x})$ is an additional iteration dependent function which influences the evaluations of infeasible solutions. The point is that the method distinguishes between feasible and infeasible individuals by adopting an additional heuristic rule (suggested earlier in (Richardson et al., 1989)): for any feasible individual $\vec{x}$ and any infeasible individual $\vec{y}$: $eval(\vec{x}) < eval(\vec{y})$, i.e., any feasible solution is better than any infeasible one.[8] In a recent study (Deb, in press), a modification to this approach is implemented with the tournament selection operator and with the following evaluation function:

$$eval(\vec{x}) = \begin{cases} f(\vec{x}), & \text{if } \vec{x} \text{ is feasible,} \\ f_{\max} + \sum_{j=1}^{m} f_j(\vec{x}), & \text{otherwise,} \end{cases}$$

where $f_{\max}$ is the function value of the worst feasible solution in the population. The main difference between this approach and Powell and Skolnick's approach is that in this approach the objective function value is not considered in evaluating an infeasible solution. Additionally, a niching scheme is introduced to maintain diversity among feasible solutions. Thus, initially the search focuses on finding feasible solutions and later when adequate number of feasible solutions are found, the algorithm finds better feasible solutions by maintaining a diversity in solutions in the feasible region. It is interesting to note that there is no need of the penalty coefficient r here, because the feasible solutions are always evaluated to be better than infeasible solutions and infeasible solutions are compared purely based on their constraint violations. However, normalization of constraints $f_j(\vec{x})$ is suggested. On a number of test problems and on an engineering design problem, this approach is better able to find constrained optimum solutions than Powell and Skolnick's approach.

The third method (Genocop III), proposed in (Michalewicz and Nazhiyath, 1995) is based on the idea of repairing infeasible individuals. Genocop III incorporates the original Genocop system, but

[8] For minimization problems.

also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points from $\mathcal{F}_l$ which satisfy linear constraints of the problem. The feasibility (in the sense of linear constraints) of these points is maintained by specialized operators. The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, i.e., they satisfy *all* constraints. Reference points $\vec{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e., $eval(\vec{r}) = f(\vec{r})$). On the other hand, search points from P_s are "repaired" for evaluation.

3.4 Methods based on decoders

Decoders offer an interesting option for all practitioners of evolutionary techniques. In these techniques a chromosome "gives instructions" on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as: "take an item if possible"—such interpretation would lead always to a feasible solution.

However, it is important to point out that several factors should be taken into account while using decoders. Each decoder imposes a mapping M between a feasible solution and decoded solution. It is important that several conditions are satisfied: (1) for each solution $s \in \mathcal{F}$ there is an encoded solution d, (2) each encoded solution d corresponds to a feasible solution s, and (3) all solutions in $\mathcal{F}$ should be represented by the same number of encodings d.[9] Additionally, it is reasonable to request that (4) the mapping M is computationally fast and (5) it has locality feature in the sense that small changes in the coded solution result in small changes in the solution itself. An interesting study on coding trees in genetic algorithm was reported in (Palmer and Kershenbaum, 1994), where the above conditions were formulated.

However, the use of decoders for continuous domains has not been investigated. Only recently (Kozieł and Michalewicz, 1998; Kozieł and Michalewicz, 1999) a new approach for solving constrained numerical optimization problems was proposed. This approach incorporates a homomorphous mapping between n-dimensional cube and a feasible search space. The mapping transforms the constrained problem at hand into unconstrained one. The method has several advantages over methods proposed earlier (no additional parameters, no need to evaluate—or penalize—infeasible solutions, easiness of approaching a solution located on the edge of the feasible region, no need for special operators, etc).

3.5 Hybrid methods

It is relatively easy to develop hybrid methods which combine evolutionary computation techniques with deterministic procedures for numerical optimization problems. In (Waagen et al. 1992) a combined an evolutionary algorithm with the direction set method of Hooke-Jeeves is described; this hybrid method was tested on three (unconstrained) test functions. In (Myung et al., 1995) the authors considered a similar approach, but they experimented with constrained problems. Again,

[9] However, as observed by Davis (1997), the requirement that all solutions in $\mathcal{F}$ should be represented by the same number of decodings seems overly strong: there are cases in which this requirement might be suboptimal. For example, suppose we have a decoding and encoding procedure which makes it impossible to represent suboptimal solutions, and which encodes the optimal one: this might be a good thing. (An example would be a graph coloring order-based chromosome, with a decoding procedure that gives each node its first legal color. This representation could not encode solutions where some nodes that could be colored were not colored, but this is a good thing!)

they combined evolutionary algorithm with some other method—developed in (Maa and Shanblatt, 1992). However, while the method of (Waagen et al. 1992) incorporated the direction set algorithm as a problem-specific operator of his evolutionary technique, in (Myung et al., 1995) the whole optimization process was divided into two separate phases.

Several other constraint handling methods deserve also some attention. For example, some methods use the values of objective function f and penalties f_j ($j = 1, \ldots, m$) as elements of a vector and apply multi-objective techniques to minimize all components of the vector. For example, in (Schaffer, 1985), Vector Evaluated Genetic Algorithm (VEGA) selects $1/(m+1)$ of the population based on each of the objectives. Such an approach was incorporated by Parmee and Purchase (1994) in the development of techniques for constrained design spaces. On the other hand, in the approach by (Surry et al., 1995), all members of the population are ranked on the basis of constraint violation. Such rank r, together with the value of the objective function f, leads to the two-objective optimization problem. This approach gave a good performance on optimization of gas supply networks.

Also, an interesting approach was reported in (Paredis, 1994). The method (described in the context of constraint satisfaction problems) is based on a co-evolutionary model, where a population of potential solutions co-evolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by fewer solutions. There is some development connected with generalizing the concept of "ant colonies" (Colorni et al., 1991) (which were originally proposed for order-based problems) to numerical domains (Bilchev and Parmee, 1995); first experiments on some test problems gave very good results (Wodrich and Bilchev, 1997). It is also possible to incorporate the knowledge of the constraints of the problem into the belief space of cultural algorithms (Reynolds, 1994); such algorithms provide a possibility of conducting an efficient search of the feasible search space (Reynolds et al., 1995).

4 Experimental results

One of the simplest and the most popular constraint-handling method is based on static penalties. In the following we define a simple static penalty method and investigate its properties using the $\mathcal{TCG}$.

The investigated static penalty method is defined as follows. For a maximization problem with the objective function f and m constraints,

$$eval(\vec{x}) = f(\vec{x}) - W \cdot v(\vec{x}),$$

where $W \geq 0$ is a constant (penalty coefficient) and function v measures the constraint violation. (Note that only one double constraint is taken into account as the $\mathcal{TCG}$ defines the feasible part of the search space as a union of all m double constraints.)

The objective function f is defined by the test-case generator (see definition of function G; Equation (14)). The constraint violation value of v for any $\vec{x}$ is defined by the following procedure:

find k such that $\vec{x} \in \mathcal{S}_k$
set $C = (2w)/\sqrt{n}$
if the whole $\mathcal{S}_k$ is infeasible **then** $v(\vec{x}) = 1$
else begin
calculate distance $Dist$ between $\vec{x}$ and the center of the subspace $\mathcal{S}_k$
if $Dist < r_1$ **then** $v(\vec{x}) = C \cdot (r_1 - Dist)$

else if $Dist > r_2$ **then** $v(\vec{x}) = C \cdot (Dist - r_2)$
else $v(\vec{x}) = 0$
end

The radii r_1 and r_2 are defined by Equations (8). Thus the constraint violation measure v returns

- 1, if the evaluated point is in infeasible subspace (i.e., subspace without a ring);
- 0, if the evaluated point is feasible;
- q, if the evaluated point is infeasible, but the corresponding subspace is partially feasible. It means that the point $\vec{x}$ is either inside the smaller ring or outside the larger one. In both cases q is a scaled distance of this point to the boundary of a closer ring. Note that the scaling factor C guarantees that $0 < q \leq 1$.

Note, that the values of the objective function for feasible points of the search space stay in the range $[0, 1]$; the value at the global optimum is always 1. Thus, both the function and constraint violation values are normalized in [0,1]. Figure 9 displays the final landscape for test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$ (Figure 8 displays the landscape before penalties are applied).

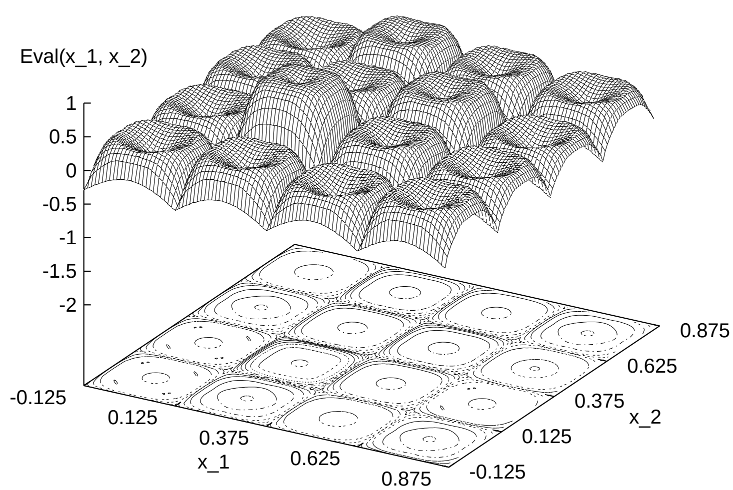


Figure 9: Final landscape for the test case $\mathcal{TCG}(2, 4, 1, \frac{1}{\sqrt{2}}, 0.8, 0)$. The penalty coefficient W is set to 1. Contours with function values ranging from -0.2 to 1.0 at a step of 0.2 are drawn at the base

To test the usefulness of the $\mathcal{TCG}$, a simple steady-state evolutionary algorithm was developed. We have used a constant population size of 100 and each individual is a vector $\vec{x}$ of n floating-point components. Parent selection was performed by a standard binary tournament selection. An offspring replaces the worst individual determined by a binary tournament. One of three operators was used in every generation (the selection of an operator was done accordingly to constant probabilities 0.5, 0.15, and 0.35, respectively):

- Gaussian mutation: $\vec{x} \leftarrow \vec{x} + N(0, \vec{\sigma})$, where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviations $\vec{\sigma}$ (in all experiments reported in this section, we have used a value of $1/(2\sqrt{n})$, which depends only on the dimensionality of the problem).

- uniform crossover: $\vec{z} \leftarrow (z_1, \ldots, z_n)$, where each z_i is either x_i or y_i (with equal probabilities), where $\vec{x}$ and $\vec{y}$ are two selected parents.
- heuristic crossover: $\vec{z} = r \cdot (\vec{x} - \vec{y}) + \vec{x}$, where r is a uniform random number between 0 and 0.25, and the parent $\vec{x}$ is not worse than $\vec{y}$.

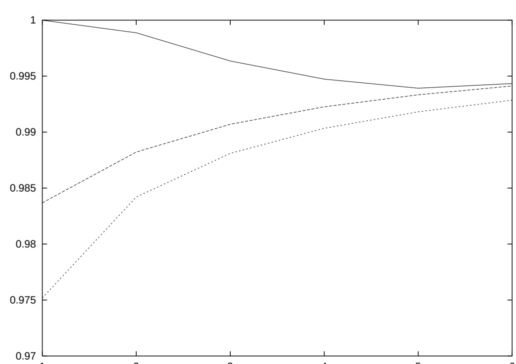
The termination condition was to quit the evolutionary loop if an improvement in the last $N = 10,000$ generations was smaller than a predefined $\epsilon = 0.001$.

As the test case generator has 6 parameters, it is difficult to fully discuss their interactions. Thus we have selected a single point from the $\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$ parameter search space:

$$n = 2;\ w = 10;\ \lambda = 1.0;\ \alpha = 0.9/\sqrt{n};\ \beta = 0.1;\ \mu = 0.1,$$

and varied one parameter at a time. In each case two figures summarize the results (which display averages of 100 runs):[10]

1. all left figures display the fitness value of the best feasible individual at the end of the run (continuous line), the average and the lowest heights among all feasible optima (broken lines).
2. all right figures rescale (linear scaling) the figure from the left: the fitness value of the best feasible individual at the end of the run (continuous line) and the average height among feasible peaks in the landscape (broken line) are displayed as fractions between 0 (which corresponds to the height of the lowest peak) and 1 (which corresponds to the height of the highest peak). Since the differences in peak heights among best few peaks are not large, the scaled peak values give a better insight into the effect of each control parameter.



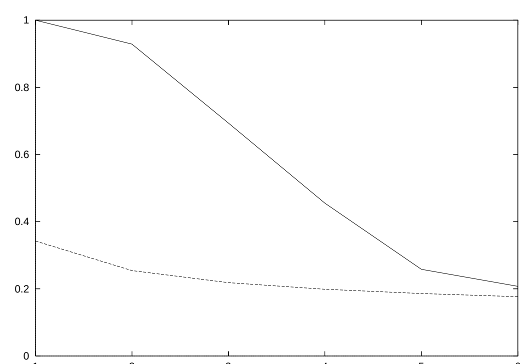


Figure 10: A run of the system for the $\mathcal{TCG}(n, 10, 1.0, 0.9/\sqrt{n}, 0.1, 0.1)$

Figure 10 displays the results of a static penalty method on the $\mathcal{TCG}$ while n is varied between 1 and 6. It is clear that an increase of n (dimensionality) reduces the efficiency of the algorithm: the value of returned solution approaches the value of the average peak height in the landscape.

Figure 11 displays the results of a static penalty method on the $\mathcal{TCG}$ while w is varied between 1 and 30 (for $w = 30$ the objective function has $w^n = 900$ peaks). An increase of w (multimodality)

[10]In all cases the values of standard deviations were similar and quite low.

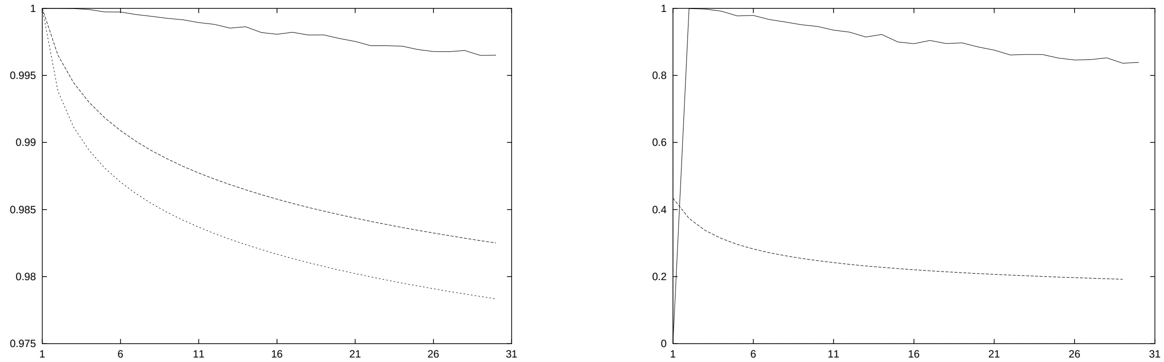


Figure 11: A run of the system for the $\mathcal{TCG}(2, w, 1.0, 0.9/\sqrt{2}, 0.1, 0.1)$

decreases the performance of the algorithm, but not to extent we have seen in the previous case (increase of n). The reason is that while $w = 10$ (Figure 10), the number of peaks grows as 10^n (for $n = 3$ there are 1000 peaks), whereas for $n = 2$ (Figure 11), the number of peaks grows as w^2 (for $w = 30$ there are 900 peaks only).

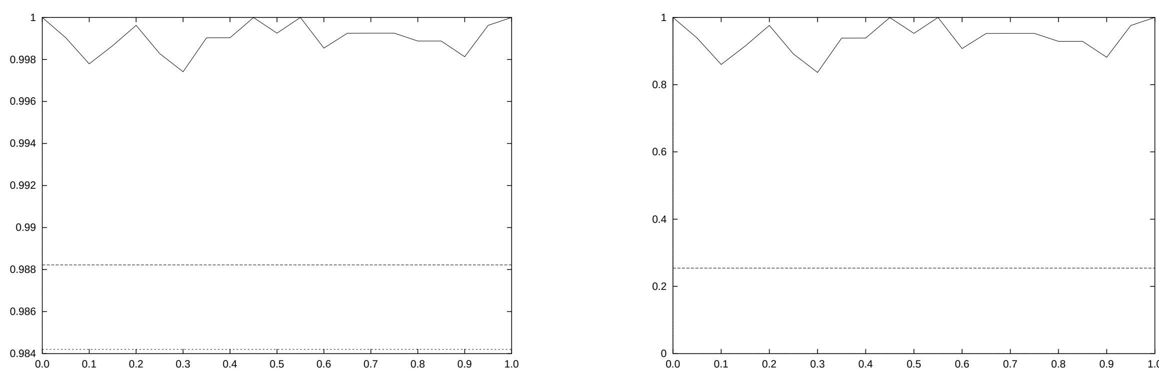


Figure 12: A run of the system for the $\mathcal{TCG}(2, 10, \lambda, 0.9/\sqrt{2}, 0.1, 0.1)$

Figure 12 displays the results of a static penalty method on the $\mathcal{TCG}$ while λ is varied between 0 and 1. It seems that the number of constraints of the test case generator does not influence the performance of the system. However, it is important to underline again that the interpretation of constraints in the $\mathcal{TCG}$ is different than usual; see a discussion in section 2.1.

Figure 13 displays the results of a static penalty method on the $\mathcal{TCG}$ while α is varied between 0 and $1/\sqrt{2}$. Clearly, larger values of α (better connectedness) improve the results of the algorithm, as it is easier to locate a feasible solution. Note an anomaly in the graph for $\alpha = 0$; this is due to the Equation (28), which provides for a different formula for a_k's when $\alpha\beta = 0$.

Figure 14 displays the results of a static penalty method on the $\mathcal{TCG}$ while β is varied between 0 and 1. Larger values of β (smaller feasibility) decrease the size of rings; however, this feature seems not to influence the performance of the algorithm significantly.

Figure 15 displays the results of a static penalty method on the $\mathcal{TCG}$ while μ is varied between 0 and 1. Higher values of μ (higher ruggedness) usually make the test case slightly harder as the heights of the peaks are changed: local maxima are of almost the same height as the global one.

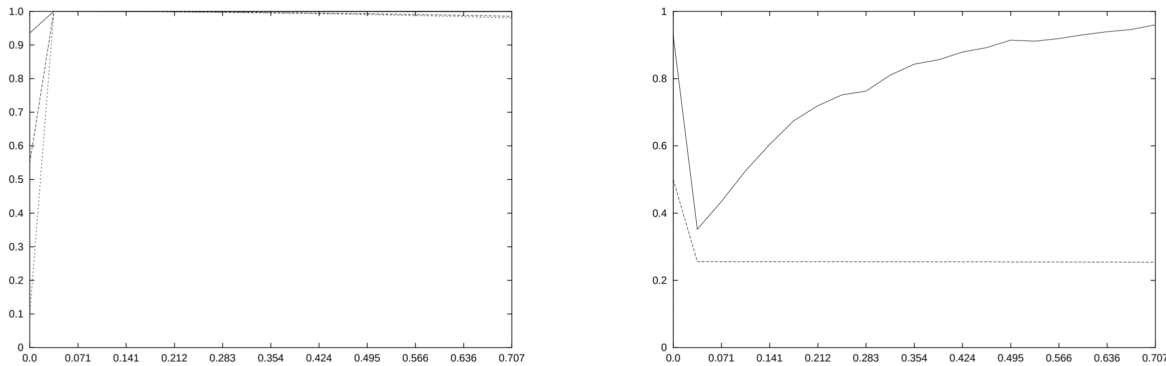


Figure 13: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, \alpha, 0.1, 0.1)$

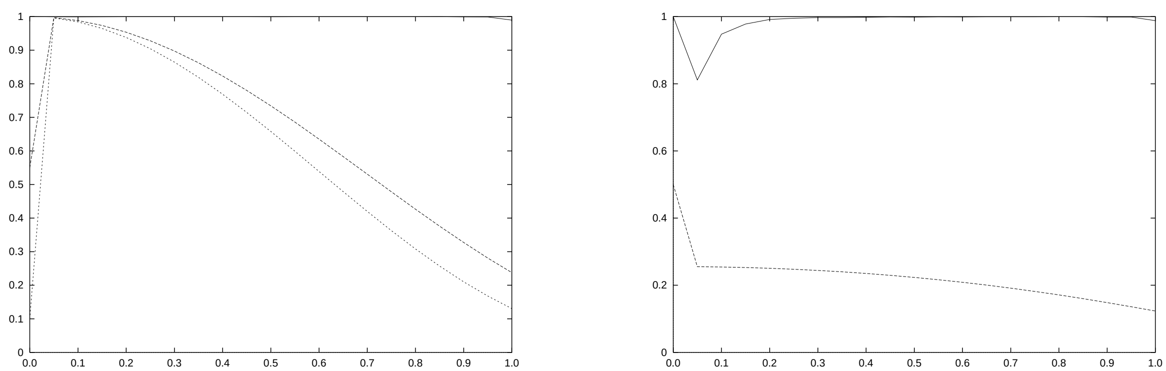


Figure 14: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, \beta, 0.1)$

We expected to see a larger influence of the parameter μ on the results, however, as the subclass of functions under investigation is relatively easy, it was not the case.

We therefore conclude, that the results obtained by an evolutionary algorithm using a static penalty approach depend mainly on the dimensionality n, the multimodality w, and the connectedness α. On the other hand, they do not significantly seem to depend on the parameters λ, β, and

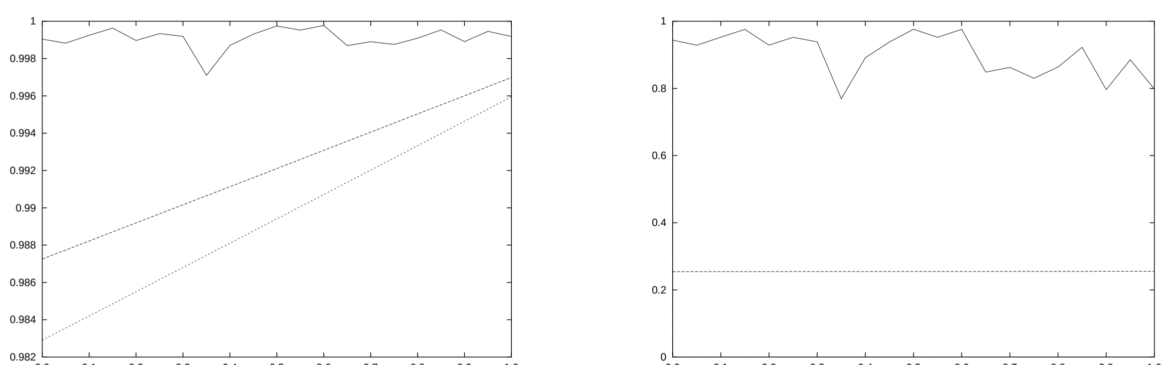


Figure 15: A run of the system for the $\mathcal{TCG}(2, 10, 1.0, 0.9/\sqrt{2}, 0.1, \mu)$

μ.

Another series of experiments aimed at exploring relationship between the value of penalty coefficient W and the results of the runs. Again, we have selected a single point from the parameter search space of $\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$:

$$n = 5;\ w = 10;\ \lambda = 0.2;\ \alpha = 0.9/\sqrt{5};\ \beta = 0.5;\ \mu = 0.1,$$

and varied penalty coefficient W from 0 to 10^4. In the first experiment we measured the ratio of feasible solutions returned by the system for different values of W; Figure 16 displays the results for $0 < W \leq 6$ (averages of 250 runs).

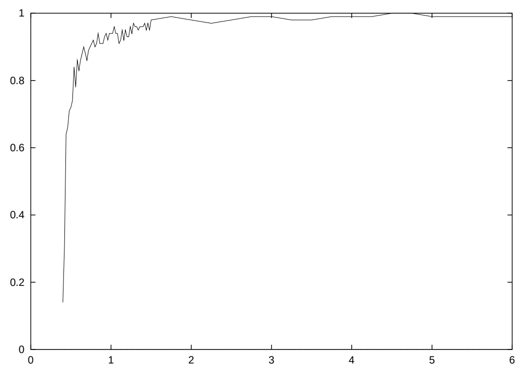


Figure 16: The success rate of the system for different values of penalty coefficient W. The success rate gives the fraction of the final solutions (out of 250 runs) which were feasible

It is interesting to note that the experiment confirmed a simple intuition: the higher penalty coefficient is, the better chance that the algorithm returns a feasible solution. For larger values of W (from 20 up to 10^4) the success rate was equal to 1 (which is the reason we did not show it in Figure 16). As infeasible solutions usually are of no interest, we conclude that high penalty coefficients guarantee that the final solution returned by the system is feasible.

However, a separate issue (apart from feasibility) is the quality of the returned solution. The next two graphs (Figure 17) clearly make the point. The left graph displays the average fitness value of the best feasible individual at the end of the run (continuous line), together with the average height and the lowest height among all feasible optima (broken lines). The right graph, on the other hand, rescales the one from the left: the fitness value of the best feasible individual at the end of the run (continuous line) and the average height among feasible peaks in the landscape (broken line) are displayed as fractions between 0 (which corresponds to the height of the lowest peak) and 1 (which corresponds to the height of the highest peak).

The (similar) graphs of Figure 17 confirm another intuition connected with static penalties: it is difficult to guess the "right" value of the penalty coefficient as different landscapes require different values of W! Note that low values of W (i.e., values below 0.4) produce poor quality results; on the other hand, for larger values of W (i.e., values larger than 1.5), the quality of solutions drops slowly (it stabilizes later — for large W — at the level of 0.95). Thus the best values of penalty coefficient W for the particular landscape of the $\mathcal{TCG}(5, 10, 0.2, 0.9/\sqrt{5}, 0.5, 0.1)$ (which has $n^w = 10^5$ local optima) are in the range $[0.6, 0.75]$.

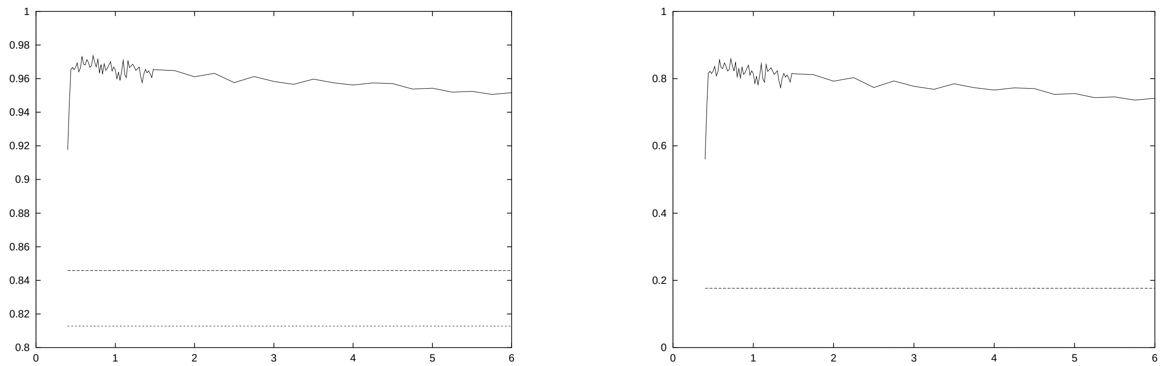


Figure 17: The quality of solution found for different values of penalty coefficient W

The results of experiments reported in Figures 16 and 17 may trigger many additional questions. For example, (1) what is the role of the selection method in these experiments? or (2) what are the merits of a dynamic penalty method as opposed to static approach? In the following, we address these two issues in turn.

In evaluating various constraint-handling techniques it is important to take into account other components of evolutionary algorithm. It is difficult to compare two constraint-handling techniques if they are incorporated in two algorithms with different selection methods, operators, parameters, etc. To illustrate the point, we have replaced the tournament selection by a proportional selection, leaving everything else in the algorithm without any change. Graphs of Figure 18 display the results: the success rate (in terms of finding a feasible solution) and the quality of solution found.

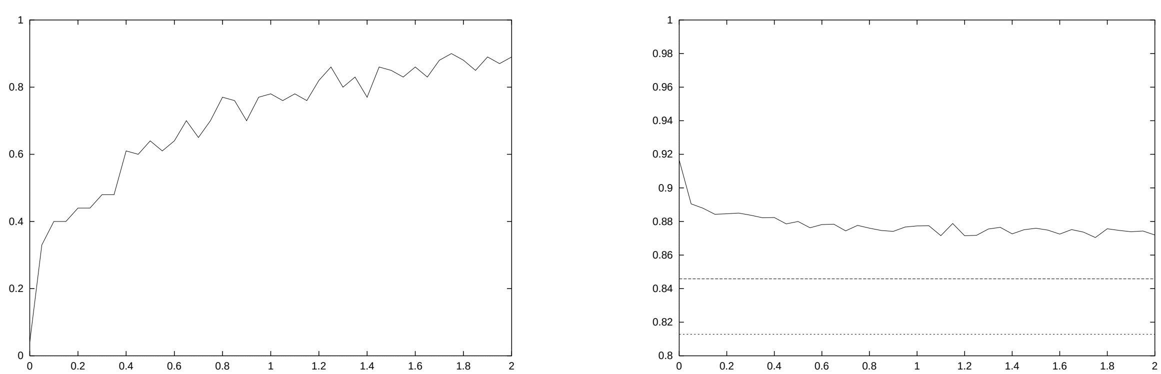


Figure 18: The success rate (left graph) and the quality of solution found (right graph) for different values of W, where a proportional selection replaced the tournament selection. Averages over 250 runs

As expected (Figure 18, left), the performance of the algorithm drops in a significant way. Proportional selection is much more sensitive to the actual values of the objective function.

We have experimented also with two dynamic penalty approaches. In both cases the maximum number of generations was set to $T = 20,000$ and

$$W_1(t) = 10 \cdot \frac{2t}{T}, \text{ and } W_2(t) = 10 \cdot \left(\frac{2t}{T}\right)^2,$$

where t is the current generation number. Thus, W_1 varies between 0 and 20 (linearly), whereas W_2 varies between 0 and 40 following a quadratic curve.

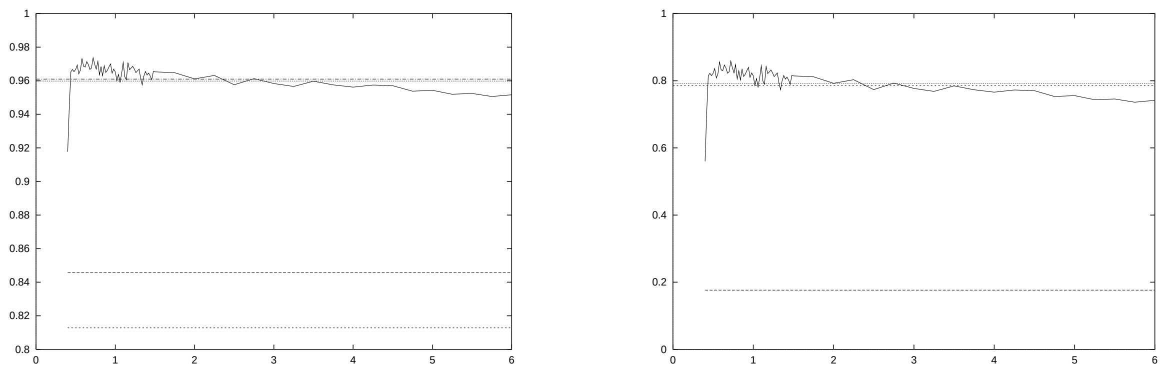


Figure 19: The quality of solution found for two dynamic penalty methods versus previously discussed static penalty approach

The results of these experiments are displayed in Figure 19. As before, the right graph rescales the left (as the value of 0 corresponds to the value of the lowest peak in the landscape). The continuous line (on both graphs) indicates the quality of a feasible solution found (average of 250 runs) for static values of W (repetition of graphs of Figure 17). The success rate for finding a feasible solution was 1 for both schemes. Two horizontal broken lines indicate the performance of the system for the two dynamic penalty approaches (W_2 being slightly better than W_1). The conclusions are clear: for most values of W (for all $W > 3$) both dynamic penalty schemes perform better than a static approach. However, it is possible to find a static value (in our case, say, $0.6 \leq W \leq 1.5$), where the static approach outperforms both dynamic schemes.

5 Summary

We have discussed briefly how the proposed test case generator $\mathcal{TCG}(n, w, \lambda, \alpha, \beta, \mu)$ can be used for evaluation of a constraint-handling technique. As explained in section 2.5, the parameter n controls the dimensionality of the test function, the parameter w controls the modality of the function, the parameter λ controls the number of constraints in the search space, the parameter α controls the connectedness of the feasible search space, the parameter β controls the ratio of the feasible to total search space, and the parameter μ controls the ruggedness of the test function.

We believe that such a constrained test problem generator should serve the purpose of testing any method for constrained parameter optimization. Moreover, one can also use the $\mathcal{TCG}$ for testing any method for unconstrained optimization (e.g., operators, selection methods, etc). In the previous section we have indicated how it can be used to evaluate merits and drawbacks of one particular constraint handling method (static penalties). Note that it is possible to analyse further the performance of a method by varying two or more parameters of the $\mathcal{TCG}$.

The proposed test case generator is far from perfect. It defines a landscape which is a collection of site-wise optimizable functions, each defined on different subspaces of equal sizes. Because of that all basins of attractions have the same size, moreover, all points at the boundary between two basins of attraction are equifitted. The local optima are located in centers of the hypercubes; all feasible regions are centered around the local optima. Note also, that while we can change the number of constraints, there is precisely one (for $\alpha beta > 0$) active constraint at the global optimum.

Some of these weaknesses can be corrected easily. For example, in order to avoid the the symmetry and equal-sized basin of attraction of all subspaces, we may modify the $\mathcal{TCG}$ in the following two ways. First, the parameter vector $\vec{x}$ may be transformed into another parameter vector $\vec{y}$ using a non-linear mapping $\vec{y} = g(\vec{x})$. The mapping function may be chosen in a way so as to have the lower and upper bounds of each parameter y_i equal to zero and one, respectively. Such a non-linear mapping will make all subspaces of different size. Second, in order to make the feasible search space asymmetric, the center of each subspace for the outer hyper-sphere can be made different from that of the inner hyper-sphere. The following update of the center for the outer hyper-sphere can be used:

$$p_i^k = \left(l_i^k + u_i^k\right)/2 + \gamma_i^k \epsilon_i^k,$$

where ϵ_i^k is a small number denoting the maximum difference allowed between centers of inner and outer hyper-spheres, and γ_i^k is a random number between zero and one. To avoid using another control parameter, ϵ_i^k can be assumed to be the same for all subspaces. This modification makes the feasible search space asymmetric, although the maximum at each subspace remains the same as in the original function.

It might be worthwhile to modify further the $\mathcal{TCG}$ to parametrize the number of active constraints at the optima. It seems necessary to introduce a possibility of a more gradual increment of the number of peaks. In the current version of the test case generator, $w = 1$ implies one peak, and $w = 2$ implies 2^n peaks (this was the reason for using low values for parameter n). Also, the difference between the lowest and the highest values of the peaks in the search space are, in the present model, too small.

All these limitations of the $\mathcal{TCG}$ diminish its significance for being a useful tool for modeling real-world problems, which may have quite different characteristics. Note also, that it is necessary to develop an additional tool, which would map a real-world problem into a particular configuration of the test case generator. In such a case, the test case generator should be able to "recommend" the most suitable constraint-handling method.

However, the proposed $\mathcal{TCG}$ is even in its current form an important tool for analysing any constraint-handling method in nonlinear programming (for any algorithm: not necessarily evolutionary algorithm), and it represents an important step in the "right" direction. The web page http://www.daimi.au.dk/ marsch/TCG.html contains two files: the $\mathcal{TCG}$ file TCG.c (standard C), and TestTCG.c: an example file that shows how to use the $\mathcal{TCG}$.

Acknowledgments

The second author acknowledges the support provided by the Alexander von Humboldt Foundation, Germany. The research reported in this paper was partially supported by the ESPRIT Project 20288 Cooperation Research in Information Technology (CRIT-2): "Evolutionary Real-time Optimization System for Ecological Power Control".

References

Bäck, T., F. Hoffmeister, and H.-P. Schwefel (1991). A survey of evolution strategies. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the* 4^{th} *International Conference on Genetic Algorithms*, pp. 2–9. Morgan Kaufmann.

Bäck, T. and M. Schütz, (1996). Intelligent Mutation Rate Control in Canonical Genetic Algorithms. In Z.W. Ras and M. Michalewicz (Eds.), *Proceedings of the 9th International Symposium on Methodologies of Intelligent Systems*, Springer-Verlag.

Bean, J. C. and A. B. Hadj-Alouane (1992). A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

Bilchev, G. and I. Parmee (1995). Ant colony search vs. genetic algorithms. Technical report, Plymouth Engineering Design Centre, University of Plymouth.

Colorni, A., M. Dorigo, and V. Maniezzo (1991). Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, Paris. MIT Press/Bradford Book.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the 3^{rd} International Conference on Genetic Algorithms*, pp. 61–69. Morgan Kaufmann.

Davis, L. (1997). Private communication.

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall.

Deb, K. (in press). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering.*

DeJong, K. (1975). *The Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Ph. D. thesis, University of Michigan, Ann Harbor. *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).

Eiben, A., P.-E. Raue, and Z. Ruttkay (1994). Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, Number 866 in LNCS, pp. 78–87. Springer Verlag.

Eiben, A. and Z. Ruttkay (1996). Self-adaptivity for Constraint Satisfaction: Learning Penalty Functions. In Proceedings of the 3rd IEEE Conference on Evolutionary Computation, IEEE Press, 1996, pp. 258–261.

Eshelman, L. and J. D. Schaffer (1993). Real-coded genetic algorithms and interval-schemata. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*, Los Altos, CA, pp. 187–202. Morgan Kaufmann.

Floudas, C. and P. Pardalos (1987). *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Volume 455 of *LNCS*. Berlin: Springer Verlag.

Gregory, J. (1995). Nonlinear Programming FAQ, Usenet sci.answers. Available at ftp://rtfm.mit.edu/pub/usenet/sci.answers/nonlinear-programming-faq.

Hadj-Alouane, A. B. and J. C. Bean (1992). A genetic algorithm for the multiple-choice integer program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan.

Himmelblau, D. (1992). *Applied Nonlinear Programming.* McGraw-Hill.

Hock, W. and K. Schittkowski (1981). *Test Examples for Nonlinear Programming Codes*, Volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag.

Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), 242–254.

Joines, J. and C. Houck (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 579–584. IEEE Press.

Keane, A.J. (1996). A Brief Comparison of Some Evolutionary Optimization Methods. In V. Rayward-Smith, I. Osman, C. Reeves and G. D. Smith (Eds.), *Modern Heuristic Search Methods*, J. Wiley, pp. 255–272.

Kelly, J. and M. Laguna (1996, April 8). Genetic Algorithms Digest. V10n16.

Kozieł, S. (1997). Evolutionary algorithms in constrained numerical optimization problems on convex spaces. Electronics and Telecommunications Quarterly, 43 (1), pp. 5–18.

Kozieł, S. and Z. Michalewicz, (1998). A decoder-based evolutionary algorithm for constrained parameter optimization problems. In *Proceedings of the* 5^{th} *Conference on Parallel Problems Solving from Nature*. Springer Verlag.

Kozieł, S. and Z. Michalewicz, (1999). Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. to appear in Evolutionary Computation.

Leriche, R. G., C. Knopf-Lenoir, and R. T. Haftka (1995). A segragated genetic algorithm for constrained structural optimization. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pp. 558–565.

Maa, C. and M. Shanblatt (1992). A two-phase optimization neural network. *IEEE Transactions on Neural Networks 3*(6), 1003–1009.

Michalewicz, Z. (1995a). Genetic algorithms, numerical optimization and constraints. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pp. 151–158. Morgan Kaufmann.

Michalewicz, Z. (1995b). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics 1*(2), 177–206.

Michalewicz, Z. (1995c). A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. *Proceedings of the* 4^{th} *Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA.

Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs*. New-York: Springer Verlag. 3rd edition.

Michalewicz, Z. and N. Attia (1994). Evolutionary optimization of constrained problems. In *Proceedings of the* 3^{rd} *Annual Conference on Evolutionary Programming*, pp. 98–108. World Scientific.

Michalewicz, Z., D. Dasgupta, R.G. Le Riche, and M. Schoenauer (1996). Test-case Generator for Constrained Parameter Optimization Techniques. Proceedings of EUROGEN'99, May 30 – June 3, 1999, Jyväskylä, Finland.

Michalewicz, Z., D. Dasgupta, R.G. Le Riche, and M. Schoenauer (1996). Evolutionary Algorithms for Constrained Engineering Problems. Computers & Industrial Engineering Journal, Vol.30, No.2.

Michalewicz, Z., Deb, K., Schmidt, M., and Stidsen, T. (1999a). Evolutionary Algorithms for Engineering Applications. Proceedings of the EUROGEN'99, May 30 – June 3, 1999, Jyväskylä, Finland.

Michalewicz, Z., Deb, K., Schmidt, M., and Stidsen, T. (1999b). Towards Understanding Constraint-Handling Methods in Evolutionary Algorithms. Submitted for publication.

Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the* 4^{th} *International Conference on Genetic Algorithms*, pp. 151–157. Morgan Kaufmann.

Michalewicz, Z., T. Logan, and S. Swaminathan (1994). Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the* 3^{rd} *Annual Conference on Evolutionary Programming*, pp. 84–97. World Scientific.

Michalewicz, Z. and M. Michalewicz (1995). Pro-Life versus Pro-Choice Strategies in Evolutionary Computation Techniques. Chapter 10 in *Evolutionary Computation*, IEEE Press.

Michalewicz, Z. and G. Nazhiyath (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In D. B. Fogel (Ed.), *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pp. 647–651. IEEE Press.

Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In P. J. Angeline and T. Bäck (Eds.), *Proceedings of the* 5^{th} *Annual Conference on Evolutionary Programming.*

Michalewicz, Z. and M. Schoenauer (1996). Evolutionary computation for constrained Parameter Optimization Problems. Evolutionary Computation, Vol.4, No.1, pp.1–32.

Mühlenbein, H. and H.-M. Voigt (1995). Gene pool recombination for the breeder genetic algorithm. In *Proceedings of the Metaheuristics International Conference*, pp. 19–25.

Myung, H., J.-H. Kim, and D. Fogel (1995). Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the* 4^{th} *Annual Conference on Evolutionary Programming*, pp. 449–463. MIT Press.

Orvosh, D. and L. Davis (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 650. Morgan Kaufmann.

Palmer, C.C. and A. Kershenbaum (1994). Representing Trees in Genetic Algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, 27–29 June 1994, 379–384.

Paredis, J. (1994). Coevolutionary constraint satisfaction. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the* 3^{rd} *Conference on Parallel Problems Solving from Nature*, pp. 46–55. Springer Verlag.

Parmee, I. and G. Purchase (1994). The development of directed genetic search technique for heavily constrained design spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, pp. 97–102. University of Plymouth.

Powell, D. and M. M. Skolnick (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 424–430. Morgan Kaufmann.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag.

Renders, J.-M. and H. Bersini (1994). Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 312–317. IEEE Press.

Reynolds, R. (1994). An introduction to cultural algorithms. In *Proceedings of the* 3^{rd} *Annual Conference on Evolutionary Programming*, pp. 131–139. World Scientific.

Reynolds, R., Z. Michalewicz, and M. Cavaretta (1995). Using cultural algorithms for constraint handling in Genocop. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the* 4^{th} *Annual Conference on Evolutionary Programming*, pp. 298–305. MIT Press.

Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Proceedings of the* 3^{rd} *International Conference on Genetic Algorithms*, pp. 191–197. Morgan Kaufmann.

Rudolph, G. (1996). Convergence properties of evolutionary algorithms. Hamburg: Verlag Dr. Kovacheckc.

Schaffer, D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the* 1^{st} *International Conference on Genetic Algorithms*. Laurence Erlbaum Associates.

Schoenauer, M. and Z. Michalewicz (1996). Evolutionary computation at the edge of feasibility. W. Ebeling, and H.-M. Voigt (Eds.), *Proceedings of the* 4^{th} *Conference on Parallel Problems Solving from Nature*, Springer Verlag.

Schoenauer, M. and Z. Michalewicz (1997). Boundary Operators for Constrained Parameter Optimization Problems. In T. Bäck (Ed.), *Proceedings of the* 7^{th} *International Conference on Genetic Algorithms*, pp.320–329. Morgan Kaufmann.

Schoenauer, M. and S. Xanthakis (1993). Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 573–580. Morgan Kaufmann.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. New-York: John Wiley & Sons. 1995 – 2^{nd} edition.

Smith, A. and D. Tate (1993). Genetic optimization using a penalty function. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 499–503. Morgan Kaufmann.

Surry, P., N. Radcliffe, and I. Boyd (1995). A multi-objective approach to constrained optimization of gas supply networks. In T. Fogarty (Ed.), *Proceedings of the AISB-95 Workshop on Evolutionary Computing*, Volume 993, pp. 166–180. Springer Verlag.

van Kemenade, C.H.M. (1998). Recombinative evolutionary search. PhD Thesis, Leiden University, Netherlands, 1998.

Waagen, D., P. Diercks, and J. McDonnell (1992). The stochastic direction set algorithm: A hybrid technique for finding function extrema. In D. B. Fogel and W. Atmar (Eds.), *Proceedings of the 1st Annual Conference on Evolutionary Programming*, pp. 35–42. Evolutionary Programming Society.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building better test functions. In L. Eshelmen (Editor), Proceedings of the 6th International Conference on Genetic Alforithms, Morgam Kaufmann, 1995.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Evaluating evolutionary algorithms. Artificial Intelligence Journal, Vol.85, August 1996, pp.245–276.

Wodrich, M. and G. Bilchev (1997). Cooperative distributed search: the ant's way. Control & Cybernetics, 26 (3).

Wright, A. (1991). Genetic algorithms for real parameter optimization. In J. G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 205–218. Morgan Kaufmann.